# NAVAL POSTGRADUATE SCHOOL
## Monterey, California


# THESIS

A PERFORMANCE MEASUREMENT
METHODOLOGY FOR SOFTWARE
MULTIPLE-BACKEND DATABASE SYSTEMS

by

James R. Vincent

June 1985

Thesis Advisor: D. K. Hsiao



Approved for public release; distribution unlimited

# NAVAL POSTGRADUATE SCHOOL
## Monterey, California


# THESIS

A PERFORMANCE MEASUREMENT
METHODOLOGY FOR SOFTWARE
MULTIPLE-BACKEND DATABASE SYSTEMS

by

James R. Vincent

June 1985

Thesis Advisor: D. K. Hsiao

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

## REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

| | |
|---|---|
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE *(and Subtitle)*: A Performance Measurement Methodology for Software Multiple-Backend Database Systems | 5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis, June 1985 |
| | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s): James R. Vincent | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93943-5100 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93943-5100 | 12. REPORT DATE: June 1985 |
| | 13. NUMBER OF PAGES: 142 |
| 14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* | 15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED |
| | 15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

benchmarking, backends, controller, performance evaluation, performance-gain, capacity-growth, database, database management sytems

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Software multiple-backend database systems provide high-performance, cost-effective database management support for systems with very large and growing databases. One backend computer, called the backend controller (or, briefly, controller), controls transaction processing of the remaining backend computers, i.e., backends, and interfaces with the host computers. These systems are designed to provide performance-gain and capacity growth

(Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S N 0102-LF-014-6601

ABSTRACT (Continued)

potential by increasing the number of backends connected to their controllers.
In this thesis we develop a comprehensive methodology for the performance evaluation of multiple-backend database systems. We also apply this methodology to develop a test-database set and test-transaction mix for the evaluation of the multi-backend database system, MBDS.

Approved for Public Release. Distribution Unlimited.

# A Performance Measurement Methodology for Software Multiple-Backend Database Systems

by

James R. Vincent
Captain, United States Air Force
B.S., Fitchburg State College, 1970
B.S., University of Utah, 1979
M.S., University of Northern Colorado, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1985

ABSTRACT

**Software multiple-backend database systems** provide high-performance, cost-effective database management support for systems with very large and growing databases. One backend computer, called the **backend controller** (or, briefly, **controller**), controls transaction processing of the remaining backend computers, i.e., **backends**, and interfaces with the host computers. These systems are designed to provide performance-gain and capacity-growth potential by increasing the number of backends connected to their controllers.

In this thesis we develop a comprehensive methodology for the performance evaluation of multiple-backend database systems. We also apply this methodology to develop a test-database set and test-transaction mix for the evaluation of the multi-backend database system, MBDS.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

## ACKNOWLEDGEMENT

This thesis is part of ongoing database systems research efforts being conducted at the Laboratory for Database Systems Research at the Naval Postgraduate School, Monterey, Ca 93943, under the direction of Dr. David K. Hsiao. This research is supported by grants from the Department of Defense STARS Program, and from the Office of Naval Research.

I would like to express my sincere appreciation to the following people:

Dr. David K. Hsiao for providing the motivation, and for his enthusiastic encouragement and support.

A hearty thanks to Steve Demurjian, a doctoral candidate in Computer Science at Ohio State University, who is currently serving as a research assistant at the Naval Postgraduate School. His keen insight, patient guidance, and contagious good humor were truly invaluable assets over the past nine months!

And finally, a very special thank-you to my wife, Pat, and my children, Chris, Suzie, and Matt, for their love, patience, faith, and understanding. I love you!

# I. INTRODUCTION

In the history of science there are two converging avenues along which flows the potential of progress: the avenue of ideas and the avenue of techniques. It is the confluence of these that has made possible the marvels of modern civilization.

. . . . . .

Of all the basic techniques perhaps none is more fundamental than that of measurement. [Ref. 1: p. 357]

## A. THE BACKGROUND

With the advent of the computer age, information processing has taken on a special significance. Most organizations now regard **information** as a valued corporate resource. Managers rely on timely, accurate information to aid them in decision making.

To satisfy this need for fast, accurate, efficient, and economical information processing, a variety of **database systems** have evolved. These systems consist of hardware components coupled with specialized software packages called **database management systems** (DBMS) [Ref. 2: p. 6]. Three different database system approaches have emerged. These include the traditional mainframe-based approach, the software single-backend approach, and the software multiple-backend approach [Ref. 3: pp. 3-5].

### 1. Traditional Mainframe-Based Database Systems

In a **traditional mainframe-based** database system, the DBMS runs on a large mainframe computer as an application program which must share the computer's resources with all of the other executing application programs. (See Figure 1.) Examples of systems based on this approach include IBM's Information Management System (IMS) and Structured English Query

Language/Data System, (SQL/DS), Relational Technology Incorporated's INGRES, Oracle's ORACLE, and Sperry Univac's DMS-1100 [Ref. 3: p. 3].

The main deficiency of this approach is that whenever the system workload increases, system performance decreases. Attempts to solve this performance problem usually involve upgrading the mainframe computer to a larger, more powerful model. But this fix is very expensive, and the additional expenses do not guarantee a proportional improvement in database system performance, since the other application programs still compete for system resources which are controlled by the mainframe computer [Ref. 3: p. 3].

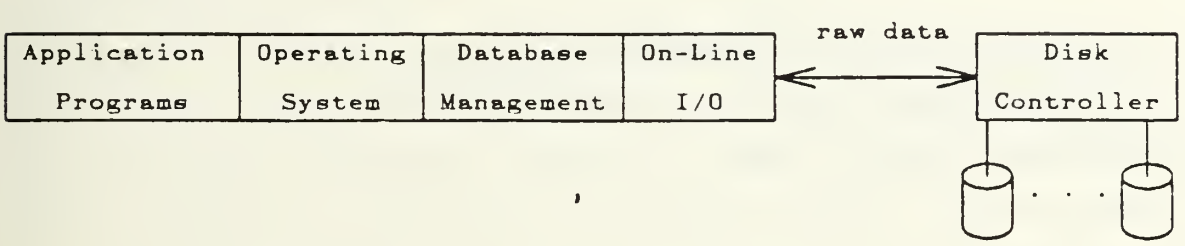


Figure 1. The Traditional Approach to Database Management.

2. Software Single-Backend Database Systems

In a **software single-backend** database system, the DBMS runs on a separate, dedicated computer system called the **backend**. (See Figure 2.) By offloading the DBMS to a backend, we free up the host mainframe computer for other tasks. Since the backend does not have to share the resources of the mainframe computer with any other processes, the result is improved database system performance. The primary example of a system based on this approach is XDMS which has been developed by Bell Laboratories. The Britton-Lee Intelligent Database Machine (IDM/500) is an example of a hardware-assisted system which makes use of the software single-backend approach by off-loading the DBMS software to a special-purpose computer. The single-backend approach is less expensive than a mainframe upgrade, but is still susceptible to performance degradation when the database system workload, (i.e., the workload of the backend), increases [Ref. 3: pp. 4-5].

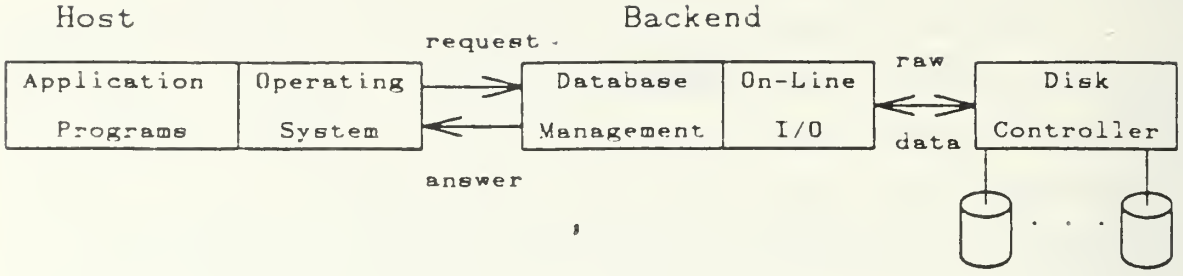


Figure 2. The Software Single-Backend Approach.

3. Software Multiple-Backend Database Systems

The **software multiple-backend approach** is more unconventional. One backend computer, called the **backend controller**, controls transaction processing of the remaining computers, or **backends**, and interfaces with the host mainframe computer. A communications bus links the controller to the backends. The backends perform the requested database operations on the database which is distributed on the backends' disk systems. (See Figure 3.) Examples of systems based on this approach include Teradata Corporation's DBC/1012 Data Base Computer System, and the Naval Postgraduate School's research system, MBDS [Ref. 3: pp. 5-6].

These systems were designed to overcome the upgrade and performance problems experienced with traditional mainframe-based and conventional software single-backend database systems. Proponents of this approach present performance-gain and capacity-growth claims based on the following design goals: (1) by increasing the number of backends while keeping the database size and size of the transaction response set constant, the system will produce a nearly reciprocal decrease in response time for the same transaction mix; (2) by increasing the number of backends in the same proportion to corresponding increases in the database size and size of the transaction response set, the response time will remain relatively constant [Ref. 3: pp. 5-6].

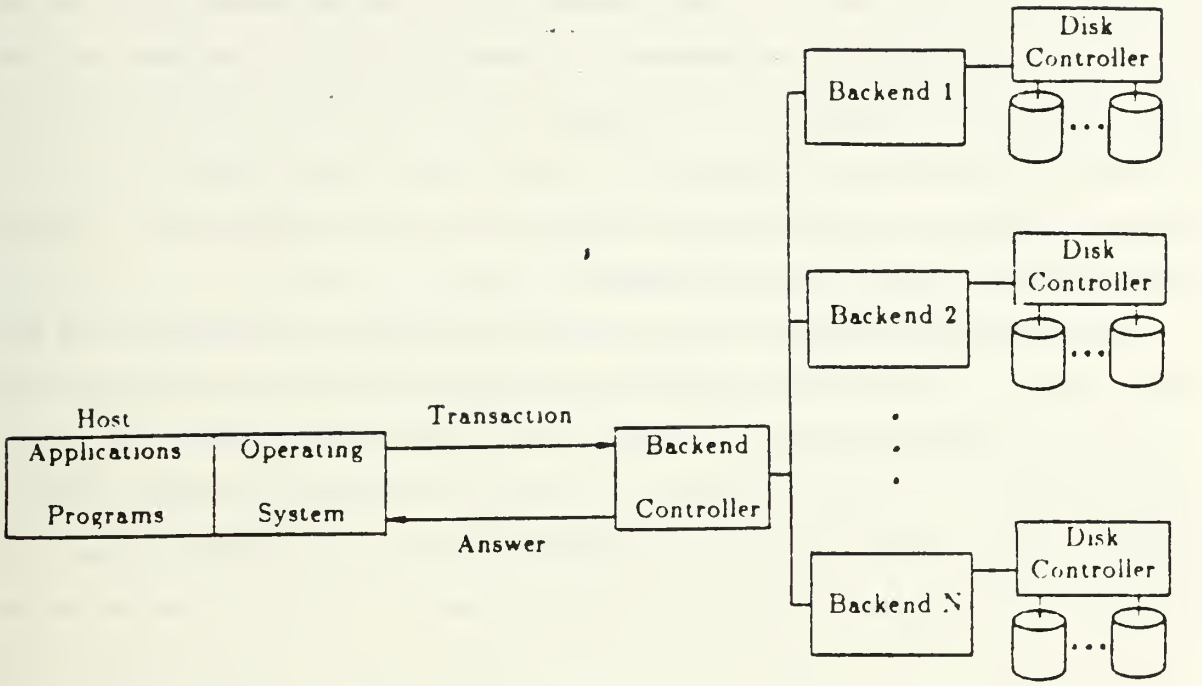


Figure 3. The Software Multiple-Backend Approach.

In [Ref. 4: pp. 12-13], Demurjian and Hsiao cite three design requirements inherent to software multi-backend database systems. First, these systems must be **expandable** by adding more backends to support both the performance-gain and capacity-growth claims. Second, the system must use readily available, "off-the-shelf" hardware to allow for ease of expansion without noticeable system interruption. Also, the software used on each backend should be designed to allow integration of one or more backends into the system by simply reproducing the DBMS software of one backend onto the new backend(s). Third, the database records must be **evenly distributed** across the backends' secondary storage devices. This will permit each backend to work in parallel with the other backends, with each concentrating on its own specific portion of the database. With these requirements, the system designers project significant potential for performance gain and capacity growth [Ref. 4: pp. 12-13].

## B. AN OVERVIEW

The basic scope of this thesis is to develop a comprehensive performance evaluation methodology for software multi-backend database systems, and to apply this methodology to the development of a test-database set and test-transaction mix for the evaluation of a specific multi-backend database system known as MBDS, which is being developed at the Naval Postgraduate School's Laboratory for Database Systems Research.

The thesis consists of two main parts. First, we present a methodology for designing a test-database set which can be used to verify the performance-gain and capacity-growth claims of the software multi-backend approach. By **test-database set**, we mean the collection of one or more databases which will be used for testing. (Each database is a collection of one or more files.) Second, we develop a test-transaction mix to verify the performance-gain and capacity-growth claims, and to measure the overall system performance of MBDS. By **test-transaction mix**, we mean the grouping of database operations, queries, or requests to be used for system testing. While the techniques we develop to evaluate MBDS performance will necessarily be system dependent, evaluators of other software multi-backend database systems may be able to use these techniques as a guideline to develop measurement strategies for evaluating their own unique systems.

This thesis provides a logical continuation of two prior projects aimed at developing a comprehensive performance measurement methodology for MBDS. Kovalchik [Ref. 5] has developed a set of performance-measurement tools for conducting system testing. These tools include a test-file record generation program to create a test database; a database load subsystem to load test files and create required directory entries; and a request generation subsystem to create, execute, and/or archive database operation requests for the test-transaction mix.

Tekampe and Watson [Ref. 6] have developed a performance measurement methodology for database systems to provide both external and internal performance measurements by embedding timing checkpoints at strategic locations in the MBDS software to provide the required measurements. Their

system also provides a set of processing flags which may be set on/off as desired to enable processing without timing measurements, with external measurements only, or with both external and internal measurements.

The external measurement facility enables us to collect performance statistics at the macroscopic level, without regard for the inner workings of the system software. This information provides a measure of the **response time** of a request, (i.e. the elapsed time expended between the user's initial request issuance and final receipt of the complete response set for the request by the user) [Ref. 7: pp. 11]. The internal measurement facility permits evaluation at the microscopic level. By observing the internal performance of the system software, we can analyze the system's work distribution. Our goal here is to be able to identify code segments which may be candidates for fine-tuning to further enhance system performance. The work done by Kovalchik, Tekampe, and Watson is further described in [Ref. 8, Ref. 9: p. 78, Ref. 10, and Ref. 11].

In developing our test database and test-transaction mix, we will follow the methodology cited by Strawser to present a general scheme which is relevant for a wide range of applications and databases. To achieve this goal, we must strive for both database-independence and application-independence [Ref. 12: pp. 11-20].

In Chapter II we analyze the performance-gain and capacity-growth claims of software multi-backend database systems to determine their influence on the design of both the test-database set and the test-transaction mix. In Chapter III we present general guidelines for test-database design which apply to all software multi-backend database systems configured with any number of backends, while in Chapter V we design the actual test database set to be used for MBDS testing. To attain database-independence, we will show how to determine database size, and how to select record sizes, number of attributes per record, and number of records per database to create a general test database set which is independent of any real application. We will also show how to select requests for the test-transaction mix to test system performance with the test database model independent of any real application. Thus, we will attain database-independence and application-independence.

The thesis will therefore present a comprehensive performance evaluation methodology for testing software multi-backend database systems configured with any number of backends, as well as a specific methodology for evaluating the research system MBDS to verify the performance-gain and capacity-growth claims. and to measure overall system performance.

## C. THE ORGANIZATION OF THE THESIS

The thesis is organized into six chapters in addition to this introduction. In Chapter II we analyze the performance-gain and capacity-growth claims to determine their influence on the test-database design, and on the design of the test-transaction mix. In Chapter III we consider pertinent database-design factors applicable to performance evaluation of all software multi-backend database systems. as well as corresponding factors influencing selection of the test-transaction mix.

Chapter IV contains an overview of the system being evaluated, the Multi-Backend Database System, MBDS. We describe the attribute-based data model, the directory structure, the five primary database operations, INSERT, DELETE, UPDATE, RETRIEVE, and RETRIEVE-COMMON, the directory and database placement, and the MBDS process structure.

In Chapter V we design the specific test-database set to use for MBDS testing, while in Chapter VI we select the requests for the test-transaction mix to evaluate the system's performance-gain and capacity-growth claims, and to test overall MBDS system performance.

Finally, we provide a summation of the thesis in Chapter VII, present conclusions, and offer suggestions for future work in performance evaluation of software multi-backend database systems in general, and MBDS in particular.

## II. THE PERFORMANCE-GAIN AND CAPACITY-GROWTH CLAIMS

In this chapter we analyze the performance-gain and capacity-growth claims to help us identify the design factors which are required for specifying a test-database set and test-transaction mix for the performance evaluation of a software multi-backend database system.

### A. GOAL (1): THE PERFORMANCE-GAIN CLAIM

To understand the inferences of goal (1) of a software multi-backend database system. first recall from Chapter I the definition of **response time** as the elapsed time between the user's initial issuance of the request, and the user's final receipt of the entire response set for the request. Goal (1) of a software multi-backend database system claims that if we maintain the same test-database set and test-transaction mix while increasing the number of backends, the database system will produce a nearly reciprocal decrease in the response time of the user's transactions [Ref. 3: p. 5]. This means that if the response time for a given transaction is X with a one backend system, then the response time would decrease to nearly X/2 with two backends, X/3 with three backends, X/4 with four backends, ..., X/m with m backends. In effect, a transaction's response time is a function of the number of backends. Therefore, goal (1) relates the number of backends **directly** to the system's **performance gains** in terms of the resulting response-time reduction [Ref. 3: pp. 5-6]. By **response-time reduction** we mean the amount of reduction in the response time of a request, when the request is processed in a system with n backends as opposed to processing the same transaction in a one backend system, while using the same test-database set [Ref. 3: p. 24]. The corresponding response-time reduction formula, as presented in [Ref. 3: p. 24], is shown in Figure 4. In this formula. configuration X represents a one backend system, while configuration Y refers to a system with m backends.

We can infer the implications of the performance-gain claim and the response-time reduction formula by analyzing Figure 5. The function R(m)

represents the response-time reduction to be realized when we increase the number of backends, m, while holding the test database and test-transaction mix constant. With these assumptions, we see that ideally R(m) = 1/m for m = 1, 2, .... n.

$$\text{The Response Time Reduction} = 100\% * \left| 1 - \left( \frac{\text{The Response Time of Configuration Y}}{\text{The Response Time of Configuration X}} \right) \right|$$

Figure 4. The Response-Time Reduction Formula.

However, the system overhead attributable to the number of backends used will inhibit our ability to realize the ideal response-time reduction curve. As shown in Figure 5, we represent the variance between the actual and ideal response-time reduction curves by the delta symbol, $\Delta$. Obviously, $\Delta(1) = 0$. As we increase the number of backends, we expect system overhead to increase. Therefore, we may infer that $\Delta(2) < \Delta(3) < \ldots < \Delta(n)$.

From this analysis we develop two logical questions. First, at what value of n will the response-time reduction stop increasing, (i.e., $R(n) \geq R(n+1)$)? Secondly, how large will n be when the system overhead becomes pronounced, (i.e., $(\Delta(n+1)/(n+1)) >> (\Delta(n)/n)$)? One of the goals of the experimental MBDS system is to determine answers to these questions via empirical performance measurements taken with different system configurations.

From this analysis, we see that the claim that the number of backends is directly related to corresponding reductions in response time may result in potentially significant performance gains for the system. To test this, we must develop a database sizing methodology which permits us to split the database into equal subsets to distribute among all of the backends, for all possible system configurations. Therefore, the performance-gain claim must be considered when designing both the test-database set and the test-transaction mix.

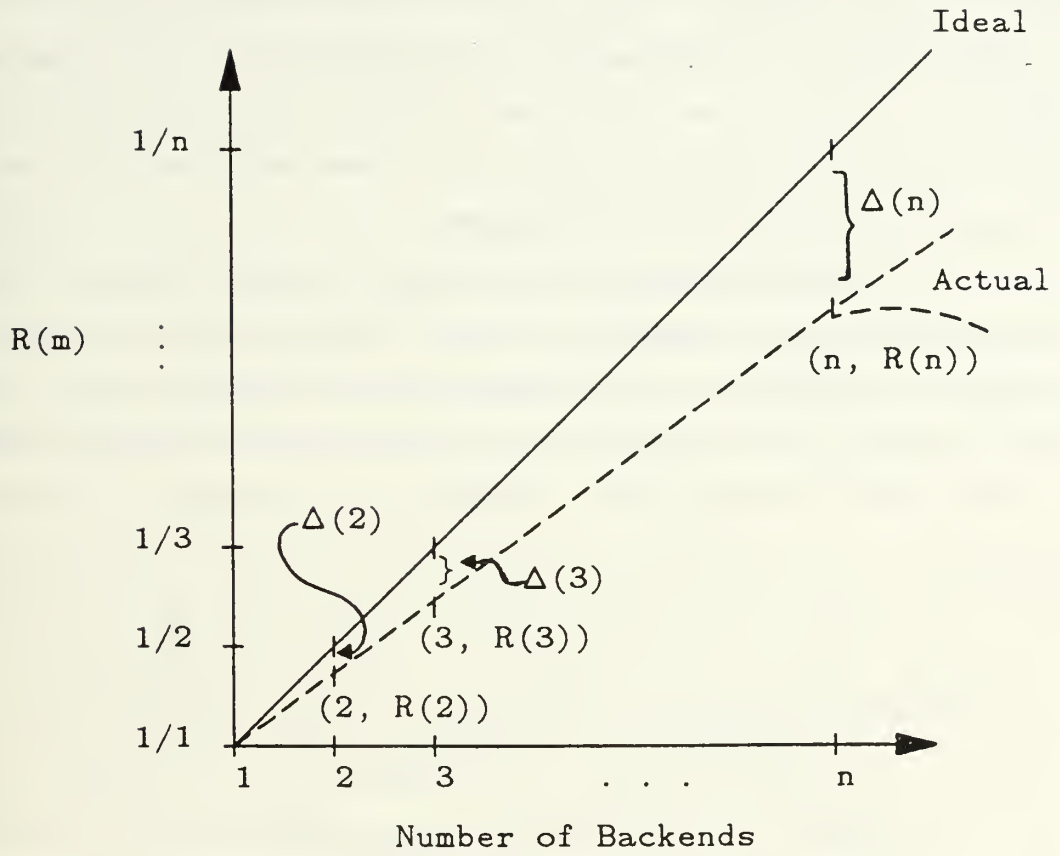


Figure 5. Response-Time Reductions Implied by Goal (1).

## B. GOAL (2): THE CAPACITY-GROWTH CLAIM

Goal (2) of a software multi-backend database system claims that if we increase the number of backends in the same proportion to corresponding increases in the size of the transaction response set, (and, therefore, the test-database size), the system will produce relatively constant response times for the same set of user transactions. (By **response set** we mean the set of responses returned by the backend(s) to the user as a result of processing a transaction.)

Now, what does goal (2) really mean? Suppose the response time for a transaction is X with a one-backend system. As the database size increases, the same test-transaction mix will eventually cause the response set size to double. Then, the claim is that the response time for the transaction in a new two

backend configuration will remain relatively constant at X. Similarly, if we expand from one to three backends, and triple the response set size, (which has also likely tripled the database size), the response time will remain nearly invariant at X. And so on. Therefore, under the capacity-growth claim, the transaction response time is a function of the number of backends, the response-set size, and, consequently, the database size.

As we see from this analysis, goal (2) directly relates the number of backends to the system's **capacity-growth** potential in terms of the resulting response-time invariance [Ref. 3: p. 6]. By **response-time invariance** we mean the amount of change in the response time of a request, when the request is processed in a one backend system with a response set of x records, as opposed to processing the same transaction in a system with m backends with a response set of mx records [Ref. 3: p. 24]. Since the size of the response set for a request is determined by the size of the database (i.e., larger databases generate more responses for the same request), the definition of response-time invariance can be restated as the amount of change in the response time of a request, when the request is processed in a one backend system with a database size of x records, as opposed to processing the same transaction in a system with m backends with a database size of mx records. The corresponding response-time invariance formula, as presented in [Ref. 3: p. 24], is shown in Figure 6. In this formula, configuration X represents a one backend system, while configuration Z represents a system with m backends, each managing x records, for a total database size of mx records.

$$\text{The Response Time Invariance} = 100\% * \left[ \left( \frac{\text{The Response Time of Configuration Z}}{\text{The Response Time of Configuration X}} \right) - 1 \right]$$

Figure 6. The Response-Time Invariance Formula.

We can infer the implications of the capacity-growth claim and the response-time invariance formula by analyzing Figure 7. The function I(m) represents the

response-time invariance to be realized when we increase the number of backends, m, while proportionally increasing the database size and the size of the transaction response set. With these assumptions, we see that ideally $I(m) = 0$ for $m = 1, 2, ..., n$.

Once again we suspect that the system overhead attributable to the number of backends used will inhibit our ability to realize the ideal response-time invariance curve. In Figure 7 we represent the variance between the actual and ideal response-time invariance curves by the delta symbol, $\Delta$. Obviously, $\Delta(1) = 0$. As we increase the number of backends, we expect system overhead to increase. Therefore, we may infer that $\Delta(2) < \Delta(3) < \ldots < \Delta(n)$.

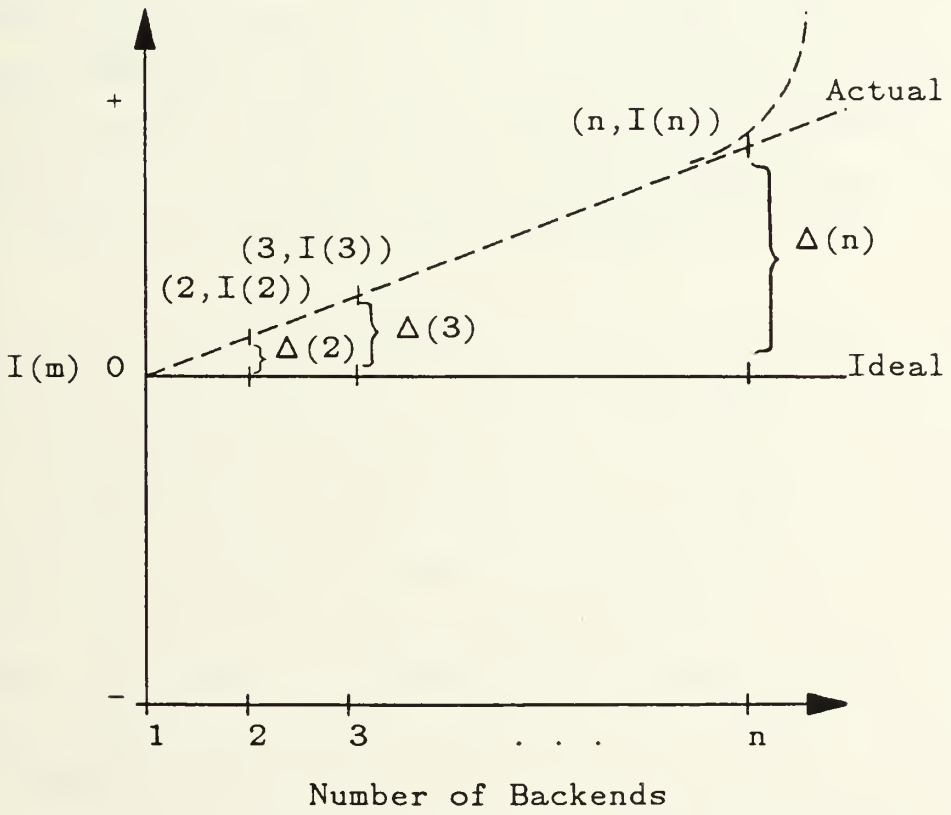


Figure 7. Response Time Invariance Implied by Goal (2).

We may again ask the question, how large will n be when the system overhead becomes pronounced, (i.e., $(\Delta(n + 1)/(n + 1)) >> ( \Delta(n)/n))$? One of the goals of the experimental MBDS system is to determine an answer to this question via empirical performance measurements taken with different system configurations.

Therefore, we see that software multi-backend database systems can produce tremendous capacity-growth potential with no degradation in performance. To test this claim, the test-database set and test-transaction mix we select must enable us to easily increase database size with corresponding increases in the response-set size. With these considerations made, let us proceed with an analysis of the factors involved in the database design and the test-transaction mix selection.

## III. THE TEST-DATABASE AND TEST-TRANSACTION DESIGN FACTORS

In this chapter we consider factors of the test-database set and the test-transaction mix applicable to a system with any number of backends. In the first section we determine the system configurations required to verify the performance-gain and capacity-growth claims. Next, we will discuss database size considerations, and then consider system-dependent database design factors. Finally, we discuss test-transaction mix considerations.

### A. SYSTEM CONFIGURATION CONSIDERATIONS

Let us consider the possible configurations for a system with M backends. Let N denote the total number of bytes in the database. Depending on the configuration being used, we must be able to evenly distribute the database to 1, 2, 3, ..., or M backends. To determine a database size which will permit an equal distribution of data to each backend in the system, we find the **least common multiple** (LCM) for the possible system configurations of 1, 2, 3, 4, ..., or M backends.

For example, consider the case where we will use four backends. The four possible system configurations are: 1, 2, 3, or 4 backends. To enable us to allocate the database in N/1, N/2, N/3, and N/4 increments, the database size must be a multiple of 12, (i.e., the LCM{1, 2, 3, 4}). If we select a database with 24,984 200-byte records, (4.8 Mbytes), the configurations listed in Table 1 are possible. We first measure performance for one backend. Then, we distribute the database evenly across two backends, three backends, and four backends, measuring the performance for each configuration. The distribution of data for the four configurations is given in Table 1. Analysis of data for this series of tests may be used to verify goal (1). That is, we collect the data to produce a graph similar to Figure 5. Table 2 summarizes the method for determining the database size (N) for a system with M backends.

TABLE 1. SAMPLE SYSTEM CONFIGURATIONS.

| Number of Backends: | Number of Mbytes per Backend: |
|---|---|
| 1 | 4.8 |
| 2 | 2.4 |
| 3 | 1.6 |
| 4 | 1.2 |

Notice that the expression for calculating the common database size multiple requires a factor of 32. The need for this factor will be explained later when we consider the record size parameter, rec-size.

TABLE 2. DATABASE-SIZE MULTIPLES.

| Maximum Number of Backends | N is a multiple of: |
|---|---|
| 1 | ( 2 x 32 x rec-size) |
| 2 | ( 2 x 32 x rec-size) |
| 3 | ( 6 x 32 x rec-size) |
| 4 | ( 12 x 32 x rec-size) |
| 5 | ( 60 x 32 x rec-size) |
| 6 | ( 60 x 32 x rec-size) |
| 7 | (420 x 32 x rec-size) |
| . . . | . . . |
| M | (LCM{1,2,...,M} x 32 x rec-size) |

KEY: M = maximum number of backends to be used.
N = total number of bytes in database.
LCM = Least Common Multiple.

Note 1: To test system claims, $M \geq 2$.
Note 2: rec-size is expressed in bytes.

Using N as determined from Table 2, we can easily summarize the database size requirements for verifying the performance-gain and capacity-growth claims. Table 3 reflects the database size parameters we require to verify goal (1), while Table 4 cites the size parameters needed to verify goal (2).

TABLE 3. DATABASE ALLOCATION TO VERIFY GOAL (1).

| Maximum Number of Backends to be used: | Portion of Database Allocated per Backend: |
|---|---|
| 1 | N |
| 2 | N/2 |
| 3 | N/3 |
| 4 | N/4 |
| . . . | . . . |
| M | N/M |

KEY. M = number of backends
N = total number of bytes in database

TABLE 4. DATABASE ALLOCATION TO VERIFY GOAL (2).

| Maximum Number of Backends | Total Database Size in Mbytes. |
|---|---|
| 1 | N |
| 2 | N/2 |
| 3 | N/3 |
| 4 | N/4 |
| . . . | . . . |
| M | N/M |

KEY. M = number of backends.
N = total number of bytes in database.

Now, let us correlate the information cited in Tables 3 and 4 to determine how many test-system configurations are required to verify the performance-gain and capacity-growth claims posed by goals (1) and (2). If we have a system with two backends, to verify goal (1) we must configure the system first with all of the database on one backend, and then with the database split evenly on two backends. To verify goal (2), we test first with all of the database on one backend, and then double the size of the database and distribute it evenly on two backends. Tables 5-8 summarize this information for systems configured with a maximum of two to five backends, respectively.

TABLE 5. TEST CONFIGURATIONS WITH TWO BACKENDS.

| Configuration Number: | Number of Backends: | Mbytes per Backend: | Total database Size in Mbytes |
|---|---|---|---|
| 1 | 1 | N | N |
| 2 | 2 | N/2 | N |
| 3 | 2 | N | 2N |

Note:
Configuration's {1,2} are required to verify goal (1).
Configuration's {1,3} are required to verify goal (2).

TABLE 6. TEST CONFIGURATIONS WITH THREE BACKENDS.

| Configuration Number: | Number of Backends: | Mbytes per Backend: | Total database Size in Mbytes |
|---|---|---|---|
| 1 | 1 | N | N |
| 2 | 2 | N/2 | N |
| 3 | 3 | N/3 | N |
| 4 | 2 | N | 2N |
| 5 | 3 | N | 3N |

Note:
Configuration's {1,2,3} are required to verify goal (1).
Configuration's {1,4,5} are required to verify goal (2).

TABLE 7. TEST CONFIGURATIONS WITH FOUR BACKENDS.

| Configuration Number: | Number of Backends: | Mbytes per Backend: | Total database Size in Mbytes |
|---|---|---|---|
| 1 | 1 | N | N |
| 2 | 2 | N/2 | N |
| 3 | 3 | N/3 | N |
| 4 | 4 | N/4 | N |
| 5 | 2 | N | 2N |
| 6 | 3 | N | 3N |
| 7 | 4 | N | 4N |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 8. TEST CONFIGURATIONS WITH FIVE BACKENDS.

| Configuration Number: | Number of Backends: | Mbytes per Backend: | Total database Size in Mbytes |
|---|---|---|---|
| 1 | 1 | N | N |
| 2 | 2 | N/2 | N |
| 3 | 3 | N/3 | N |
| 4 | 4 | N/4 | N |
| 5 | 5 | N/5 | N |
| 6 | 2 | N | 2N |
| 7 | 3 | N | 3N |
| 8 | 4 | N | 4N |
| 9 | 5 | N | 5N |

Note:
Configuration's {1,2,3,4,5} are required to verify goal (1).
Configuration's {1,6,7,8,9} are required to verify goal (2).

In general, when we have a system which is configurable with a maximum of M backends, then the number of test configurations required to verify goal (1) is M, and the number of test configurations required to verify goal (2) is (M-1), thereby making the total number of test configurations required to test a system with M backends equal to (2M - 1), i.e., (M + (M - 1)). Therefore, a system with two backends requires three test configurations; a system with three backends requires five test configurations; a system with seven backends requires thirteen configurations; etc. Using this methodology, a system evaluator may determine the number of required test configurations to verify the performance-gain and capacity-growth claims for a system with any number of backends.

## B. DATABASE-SIZE CONSIDERATIONS

Next, we consider how to determine the database-size parameter, N. To adequately measure the performance characteristics of a software multi-backend database system, we propose that three different database sizes be selected. One size should represent a small database, one size should represent a large database, and the third should represent an intermediate size between the largest and smallest values picked.

The database sizes selected may be hardware dependent. Therefore, we propose the following scheme which may be easily applied to any hardware configuration. First, the largest database size will be approximately the

maximum formated capacity, (in Mbytes), of a backend's secondary storage system. We shall call this database size N. The smallest database size will be N/4, while the intermediate size will be N/2.

As an example, assume that each backend has a single disk drive with a maximum formated capacity of 400 Mbytes. Then the maximum value of N is 400 Mbytes. However, recall that N must be a multiple of (LCM{1,2,...,M} x 32 x rec-size), where M is the maximum number of backends to be configured in our system. To continue the example, assume that we have a system with a maximum of four backends, (and therefore, four disk drives). From Table 2 we see that N must therefore be divisible by (12 x 32 x rec-size). Although we have yet to consider the record-size (parameter) value, this requirement implies that the **largest** database we may use will be divisible by 12 x 32 = 384. Therefore, the upper-bound value for N will be 399.999744 Mbytes.

If we want to use three database sizes for system testing, with the largest being 399.999744 Mbytes, then we would have the following:

| | | | | |
|---|---|---|---|---|
| N/4 | = | (399.999744 Mbytes)/4 | = | 99.999936 Mbytes |
| N/2 | = | (399.999744 Mbytes)/2 | = | 199.999872 Mbytes |
| N | = | | = | 399.999744 Mbytes. |

However, since the database size, N, is a multiple of ([the LCM{1,2,3,...,M}] x 32 x rec-size), we must consider **record size** before selecting the final value for N. Strawser notes that record-size selection is hardware specific, since it depends on the size of the unit of data management used by the particular system's architecture [Ref. 12: pp. 16-17].

For example, suppose the disk track size is 4 Kbytes. Using Strawser's scheme for dividing this 4-Kbyte track into four record sizes, we may select sizes of 2000, 1000, 400, and 200 bytes per record, resulting in a range of 2 to 20 records per track. For a system which supports a 16-Kbyte track size, we may select record sizes of 4000, 2000, 800, and 400 bytes per record, which results in a range of 4 to 40 records per track.

The key to record-size selection is to ensure that one record size is large and one small, with the other two record sizes representing intermediate values between the largest and smallest values picked. This will enable us to contrast performance for cases where there are many small records per track to cases where there are a few large records per track [Ref. 12: p. 17]. In addition, we require that the three smaller record sizes be evenly divisible into the largest record size, since this simplifies the process of determining database size. With this requirement, we may concentrate on sizing the database for the largest record size, and be assured that the selected database will accommodate the smaller record sizes as well. Since track sizes differ for various disk installations, each system evaluator must determine unique record sizes which will be compatible with the specific system's unit of data access and storage.

Assume that we decide to use a 4-Kbyte track size, with record sizes of 2000, 1000, 400, and 200 bytes per record. We will use this assumption to continue the development of a test-database set for a system with a maximum of four backends.

We can now determine the required database multiple for our example application, as follows:

$$(12 \times 32 \times 2000) \quad = \quad 768{,}000.$$

Therefore, N will be the largest multiple of 768,000 bytes which is less than 399.999744 Mbytes, (i.e., 399,999.744 bytes). Consequently, we calculate that N equals 399.36 Mbytes. Therefore, we have:

$$N/4 \quad = \quad 99.84 \text{ Mbytes}$$

$$N/2 \quad = \quad 199.68 \text{ Mbytes}$$

$$N \quad = \quad 399.36 \text{ Mbytes}.$$

Now, to ensure that these database sizes are feasible, we substitute the values of N/4, N/2, and N into Table 7 to derive Tables 9-11.

TABLE 9. FOUR BACKENDS WITH SMALL DATABASE (N/4 = 99.84 MBYTES).

| Configuration Number: | Number of Backends: | Mbytes per Backend: | Total database Size in Mbytes |
|---|---|---|---|
| 1 | 1 | 99.84 | 99.84 |
| 2 | 2 | 49.92 | 99.84 |
| 3 | 3 | 33.28 | 99.84 |
| 4 | 4 | 24.96 | 99.84 |
| 5 | 2 | 99.84 | 199.68 |
| 6 | 3 | 99.84 | 299.52 |
| 7 | 4 | 99.84 | 399.36 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 10. FOUR BACKENDS WITH MEDIUM DATABASE (N/2 = 199.68 MBYTES).

| Configuration Number: | Number of Backends: | Mbytes per Backend: | Total database Size in Mbytes |
|---|---|---|---|
| 1 | 1 | 199.68 | 199.68 |
| 2 | 2 | 99.84 | 199.68 |
| 3 | 3 | 66.56 | 199.68 |
| 4 | 4 | 49.92 | 199.68 |
| 5 | 2 | 199.68 | 399.36 |
| 6 | 3 | 199.68 | 599.04 |
| 7 | 4 | 199.68 | 798.72 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 11. FOUR BACKENDS WITH LARGE DATABASE (N = 399.36 MBYTES).

| Configuration Number: | Number of Backends: | Mbytes per Backend: | Total database Size in Mbytes |
|---|---|---|---|
| 1 | 1 | 399.36 | 399.36 |
| 2 | 2 | 199.68 | 399.36 |
| 3 | 3 | 133.12 | 399.36 |
| 4 | 4 | 99.84 | 399.36 |
| 5 | 2 | 399.36 | 798.72 |
| 6 | 3 | 399.36 | 1,198.08 |
| 7 | 4 | 399.36 | 1,597.44 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

Tables 9-11 show that the three proposed test-database sets are feasible, since they permit each database to be split evenly as required for all of the necessary test configurations. Next, we consider how to format the test-database sets. Two options seem feasible. We may use only one record type per database, or we may include all four record types in a single database.

First, consider the case where we use only one record type per database. Since we have four record sizes, we must create four separate databases, (one for each record size). Since we also want to test with three different databases, (small, medium, and large), we therefore have 12, (i.e., 3 x 4), different database configurations to be used for testing. We first consider the case for a small database, with N/4 = 99.84 Mbytes. With four record sizes, we calculate the following:

| | | |
|---|---|---|
| (99.84 Mbytes/2000 bytes) | = | 49,920 records/database |
| (99.84 Mbytes/1000 bytes) | = | 99,840 records/database |
| (99.84 Mbytes/400 bytes) | = | 249,600 records/database |
| (99.84 Mbytes/200 bytes) | = | 499,200 records/database. |

Similar calculations are done for N/2 = 199.68 Mbytes and N = 399.36 Mbytes. When we transcribe this information to Tables 9-11, we end up with the system configurations listed in Tables 12-23 below. Tables 12-15 reflect the required configurations for testing with a small database, (N/4 = 99.84 Mbytes), for each of the four record sizes. Tables 16-19 reflect the same breakout for a medium size database, (N/2 = 199.68 Mbytes), while Tables 20-23 are for a large database, (N = 399.36 Mbytes).

Tables 12-23 verify that the four record sizes we have selected are compatible with the chosen database-size sets, since they permit each database to be split evenly as required for all of the necessary test configurations. Furthermore, note that each breakout in Tables 12-23 requires 7 system configurations to verify system goals (1) and (2) for a system with a maximum of 4 backends. This implies that the performance measurement tests must be run 84 times, (i.e., 12 x 7)! Since four different record sizes are used over three database sizes, this

TABLE 12. FORMAT FOR SMALL DATABASE WITH 2000-BYTE RECORDS.

| Configuration Number: | Number of Backends: | Number of Records per Backend | Total Database Size in Records |
|---|---|---|---|
| 1 | 1 | 49,920 | 49,920 |
| 2 | 2 | 24,960 | 49,920 |
| 3 | 3 | 16,640 | 49,920 |
| 4 | 4 | 12,480 | 49,920 |
| 5 | 2 | 49,920 | 99,840 |
| 6 | 3 | 49,920 | 149,760 |
| 7 | 4 | 49,920 | 199,680 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 13. FORMAT FOR SMALL DATABASE WITH 1000-BYTE RECORDS.

| Configuration Number: | Number of Backends: | Number of Records per Backend: | Total Database Size in Records |
|---|---|---|---|
| 1 | 1 | 99,840 | 99,840 |
| 2 | 2 | 49,920 | 99,840 |
| 3 | 3 | 33,280 | 99,840 |
| 4 | 4 | 24,960 | 99,840 |
| 5 | 2 | 99,840 | 199,680 |
| 6 | 3 | 99,840 | 299,520 |
| 7 | 4 | 99,840 | 399,360 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 14. FORMAT FOR SMALL DATABASE WITH 400-BYTE RECORDS.

| Configuration Number: | Number of Backends: | Number of Records per Backend: | Total Database Size in Records |
|---|---|---|---|
| 1 | 1 | 249,600 | 249,600 |
| 2 | 2 | 124,800 | 249,600 |
| 3 | 3 | 83,200 | 249,600 |
| 4 | 4 | 62,400 | 249,600 |
| 5 | 2 | 249,600 | 499,200 |
| 6 | 3 | 249,600 | 748,800 |
| 7 | 4 | 249,600 | 998,400 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 15. FORMAT FOR SMALL DATABASE WITH 200-BYTE RECORDS.

| Configuration Number: | Number of Backends: | Number of Records per Backend: | Total Database Size in Records |
|---|---|---|---|
| 1 | 1 | 499,200 | 499,200 |
| 2 | 2 | 249,600 | 499,200 |
| 3 | 3 | 166,400 | 499,200 |
| 4 | 4 | 124,800 | 499,200 |
| 5 | 2 | 499,200 | 998,400 |
| 6 | 3 | 499,200 | 1,497,600 |
| 7 | 4 | 499,200 | 1,996,800 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 16. FORMAT FOR MEDIUM DATABASE WITH 2000-BYTE RECORDS.

| Configuration Number: | Number of Backends: | Number of Records per Backend: | Total Database Size in Records |
|---|---|---|---|
| 1 | 1 | 99,840 | 99,840 |
| 2 | 2 | 49,920 | 99,840 |
| 3 | 3 | 33,280 | 99,840 |
| 4 | 4 | 24,960 | 99,840 |
| 5 | 2 | 99,840 | 199,680 |
| 6 | 3 | 99,840 | 299,520 |
| 7 | 4 | 99,840 | 399,360 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 17. FORMAT FOR MEDIUM DATABASE WITH 1000-BYTE RECORDS.

| Configuration Number: | Number of Backends: | Number of Records per Backend: | Total Database Size in Records |
|---|---|---|---|
| 1 | 1 | 199,680 | 199,680 |
| 2 | 2 | 99,840 | 199,680 |
| 3 | 3 | 66,560 | 199,680 |
| 4 | 4 | 49,920 | 199,680 |
| 5 | 2 | 199,680 | 399,360 |
| 6 | 3 | 199,680 | 599,040 |
| 7 | 4 | 199,680 | 798,720 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 18. FORMAT FOR MEDIUM DATABASE WITH 400-BYTE RECORDS.

| Configuration Number: | Number of Backends: | Number of Records per Backend: | Total Database Size in Records |
|---|---|---|---|
| 1 | 1 | 499,200 | 499,200 |
| 2 | 2 | 249,600 | 499,200 |
| 3 | 3 | 166,400 | 499,200 |
| 4 | 4 | 124,800 | 499,200 |
| 5 | 2 | 499,200 | 998,400 |
| 6 | 3 | 499,200 | 1,497,600 |
| 7 | 4 | 499,200 | 1,996,800 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 19. FORMAT FOR MEDIUM DATABASE WITH 200-BYTE RECORDS.

| Configuration Number: | Number of Backends: | Number of Records per Backend: | Total Database Size in Records |
|---|---|---|---|
| 1 | 1 | 998,400 | 998,400 |
| 2 | 2 | 499,200 | 998,400 |
| 3 | 3 | 332,800 | 998,400 |
| 4 | 4 | 249,600 | 998,400 |
| 5 | 2 | 998,400 | 1,996,800 |
| 6 | 3 | 998,400 | 2,995,200 |
| 7 | 4 | 998,400 | 3,993,600 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 20. FORMAT FOR LARGE DATABASE WITH 2000-BYTE RECORDS.

| Configuration Number: | Number of Backends: | Number of Records per Backend: | Total Database Size in Records |
|---|---|---|---|
| 1 | 1 | 199,680 | 199,680 |
| 2 | 2 | 99,840 | 199,680 |
| 3 | 3 | 66,560 | 199,680 |
| 4 | 4 | 49,920 | 199,680 |
| 5 | 2 | 199,680 | 399,360 |
| 6 | 3 | 199,680 | 599,040 |
| 7 | 4 | 199,680 | 798,720 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 21. FORMAT FOR LARGE DATABASE WITH 1000-BYTE RECORDS.

| Configuration Number: | Number of Backends: | Number of Records per Backend: | Total Database Size in Records |
|---|---|---|---|
| 1 | 1 | 399,360 | 399,360 |
| 2 | 2 | 199,680 | 399,360 |
| 3 | 3 | 133,120 | 399,360 |
| 4 | 4 | 99,840 | 399,360 |
| 5 | 2 | 399,360 | 798,720 |
| 6 | 3 | 399,360 | 1,198,080 |
| 7 | 4 | 399,360 | 1,597,440 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 22. FORMAT FOR LARGE DATABASE WITH 400-BYTE RECORDS.

| Configuration Number: | Number of Backends: | Number of Records per Backend: | Total Database Size in Records |
|---|---|---|---|
| 1 | 1 | 998,400 | 998,400 |
| 2 | 2 | 499,200 | 998,400 |
| 3 | 3 | 332,800 | 998,400 |
| 4 | 4 | 249,600 | 998,400 |
| 5 | 2 | 998,400 | 1,996,800 |
| 6 | 3 | 998,400 | 2,995,200 |
| 7 | 4 | 998,400 | 3,993,600 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

TABLE 23. FORMAT FOR LARGE DATABASE WITH 200-BYTE RECORDS.

| Configuration Number: | Number of Backends: | Number of Records per Backend: | Total Database Size in Records |
|---|---|---|---|
| 1 | 1 | 1,996,800 | 1,996,800 |
| 2 | 2 | 998,400 | 1,996,800 |
| 3 | 3 | 665,600 | 1,996,800 |
| 4 | 4 | 499,200 | 1,996,800 |
| 5 | 2 | 1,996,800 | 3,993,600 |
| 6 | 3 | 1,996,800 | 5,990,400 |
| 7 | 4 | 1,996,800 | 7,987,200 |

Note:
Configuration's {1,2,3,4} are required to verify goal (1).
Configuration's {1,5,6,7} are required to verify goal (2).

means that twelve different sets of test-transactions will be required, one for each test database.

Next, consider the case where we have a system which will let us use four different record types in a single database. In this case, we will require just three test databases instead of twelve, since each database will contain four record types. However, each set of test-transactions will be four times as large, since they will include all of the transactions for each of the four record types. Previously, there was a separate, smaller set of test transactions for each record type and database size. Now, we require just three sets of test transactions, one for each database size. Thus, the amount of testing required will be essentially the same as that required above.

Note that if we assume that the formated disk capacity for a system is to remain the same between these two cases, then far fewer records per record type will be available for processing when all four record types are present in a single database. This database configuration may be easier to use for testing, since only 21 sets, (i.e., 3 x 7), of performance measurement tests need to be run. Each set of test-transactions will be larger, since they must include transactions for all four record types. However, the response-set size for the test-transaction mix will be smaller. (As all four record types will be distributed over the available secondary storage, fewer records per record type are stored).

Because the available secondary storage will now be shared with all four record types, we must consider how to distribute the database between the four possible record sizes. One option would be to use an equal number of records per record size. The disadvantage of this approach is that the resultant database distribution is inequitable. For example, suppose we decide to construct database containing four different record sizes, with 100 records of each size in the database. Using 2000, 1000, 400 and 200 bytes per record, this example gives us the database distribution shown in Table 24.

Table 24 shows that this design results in the 200-byte record category representing only 5.6 percent of the total database, whereas the 2000-byte record category dominates with 55.5 percent of the total database. Thus, this distribution is unfair since the record sizes themselves are unequal.

TABLE 24. SAMPLE DATABASE WITH FOUR RECORD GROUPINGS.

| Record Size in Bytes | Number of Records | Total Number of Bytes | Percent of Total Database |
|---|---|---|---|
| 2000 | 100 | 200,000 | 55.5 |
| 1000 | 100 | 100,000 | 27.8 |
| 400 | 100 | 40,000 | 11.1 |
| 200 | 100 | 20,000 | 5.6 |

Table 24 shows that this design results in the 200-byte record category representing only 5.6 percent of the total database, whereas the 2000-byte record category dominates with 55.5 percent of the total database. Thus, this distribution is unfair since the record sizes themselves are unequal.

A more equitable design would be to split the database into four equal groupings, with each quarter of the database corresponding to one of the four record-size categories. We apply this technique to our hypothetical four backend system to demonstrate its application. First, consider the small database, with N/4 = 99.84 Mbytes. Then, (99.84 Mbytes)/4 = 24.96 Mbytes per record grouping. Therefore, we have:

(24.96 Mbytes)/(2000 bytes/record) = 12,480 records

(24.96 Mbytes)/(1000 bytes/record) = 24,960 records

(24.96 Mbytes)/(400 bytes/record) = 62,400 records

(24.96 Mbytes)/(200 bytes/record) = 124,800 records.

Following through with similar calculations for N/2 and N, we can derive Tables 25-27 for the situation where we have small, medium, and large databases consisting of four record groupings per database, with records of 2000, 1000, 400, and 200 bytes per record. Once again, we see from Tables 25-27 that the database design permits each database to be split evenly and fairly as required for all of the necessary test configurations.

TABLE 25. TEST CONFIGURATIONS FOR SMALL DATABASE.

| Configuration Number | Number of Backends | Record Size in Bytes | Number of Records per Backend | Mbytes per Backend | Database Size in Mbytes |
|---|---|---|---|---|---|
| 1 | 1 | 2000 | 12,480 | 24.960 | |
| | | 1000 | 24,960 | 24.960 | |
| | | 400 | 62,400 | 24.960 | 99.84 |
| | | 200 | 124,800 | 24.960 | |
| 2 | 2 | 2000 | 6,240 | 12.480 | |
| | | 1000 | 12,480 | 12.480 | |
| | | 400 | 31,200 | 12.480 | 99.84 |
| | | 200 | 62,400 | 12.480 | |
| 3 | 3 | 2000 | 4,160 | 8.320 | |
| | | 1000 | 8,320 | 8.320 | |
| | | 400 | 20,800 | 8.320 | 99.84 |
| | | 200 | 41,600 | 8.320 | |
| 4 | 4 | 2000 | 3,120 | 6.240 | |
| | | 1000 | 6,240 | 6.240 | |
| | | 400 | 15,600 | 6.240 | 99.84 |
| | | 200 | 31,200 | 6.240 | |
| 5 | 2 | 2000 | 12,480 | 24.960 | |
| | | 1000 | 24,960 | 24.960 | |
| | | 400 | 62,400 | 24.960 | 199.68 |
| | | 200 | 124,800 | 24.960 | |
| 6 | 3 | 2000 | 12,480 | 24.960 | |
| | | 1000 | 24,960 | 24.960 | |
| | | 400 | 62,400 | 24.960 | 299.52 |
| | | 200 | 124,800 | 24.960 | |
| 7 | 4 | 2000 | 12,480 | 24.960 | |
| | | 1000 | 24,960 | 24.960 | |
| | | 400 | 62,400 | 24.960 | 399.36 |
| | | 200 | 124,800 | 24.960 | |

We may now explain the requirement for the multiple of 32 in the database-size relation of Table 2. First, recall that, in general, N is a multiple of (LCM{1,2,....M} x 32 x rec-size). In our methodology for selecting a small, medium, or large database, we decided to select database-size increments of N/4, N/2, and N. Thus, N is a multiple of 1, 2, and 4. Since the LCM{1,2,4} is 4, then N must be divisible by 4. Secondly, to enable us to handle the four record-size groupings in a single database, we must be able to split the database into four even quarters. The worse case is when we take the small-size database, (N/4), and split it into quarters, one-quarter for each of four record sizes. Therefore, N, (which we already know must be divisible by 4), must be divisible by 4 again to be split into quarters. But, (N/4)/4 is the same as N/16. The effect is to require that the total database size, N, be divisible by 16.

TABLE 26. TEST CONFIGURATIONS FOR MEDIUM DATABASE.

| Configuration Number | Number of Backends | Record Size in Bytes | Number of Records per Backend | Mbytes per Backend | Database Size in Mbytes |
|---|---|---|---|---|---|
| 1 | 1 | 2000 | 24,960 | 49.92 | |
| | | 1000 | 49,920 | 49.92 | |
| | | 400 | 124,800 | 49.92 | 199.68 |
| | | 200 | 249,600 | 49.92 | |
| 2 | 2 | 2000 | 12,480 | 24.96 | |
| | | 1000 | 24,960 | 24.96 | |
| | | 400 | 62,400 | 24.96 | 199.68 |
| | | 200 | 124,800 | 24.96 | |
| 3 | 3 | 2000 | 8,320 | 16.64 | |
| | | 1000 | 16,640 | 16.64 | |
| | | 400 | 41,600 | 16.64 | 199.68 |
| | | 200 | 83,200 | 16.64 | |
| 4 | 4 | 2000 | 6,240 | 12.48 | |
| | | 1000 | 12,480 | 12.48 | |
| | | 400 | 31,200 | 12.48 | 199.68 |
| | | 200 | 62,400 | 12.48 | |
| 5 | 2 | 2000 | 24,960 | 49.92 | |
| | | 1000 | 49,920 | 49.92 | |
| | | 400 | 124,800 | 49.92 | 399.36 |
| | | 200 | 249,600 | 49.92 | |
| 6 | 3 | 2000 | 24,960 | 49.92 | |
| | | 1000 | 49,920 | 49.92 | |
| | | 400 | 124,800 | 49.92 | 599.04 |
| | | 200 | 249,600 | 49.92 | |
| 7 | 4 | 2000 | 24,960 | 49.92 | |
| | | 1000 | 49,920 | 49.92 | |
| | | 400 | 124,800 | 49.92 | 798.72 |
| | | 200 | 249,600 | 49.92 | |

Finally, we require that the database represented by N/16 also be divisible by 2. This final requirement is actually related to the MBDS storage mechanism, which stores records into clusters, as we shall discuss in Chapters IV and V. By requiring that the database be divisible by this final factor of 2, we make it possible for each MBDS cluster to hold an even number of records. Note that this factor of 2 is not a general requirement for all software multi-backend database systems. In fact, we can design test databases for MBDS without this requirement. However, the design and formating of the MBDS test-database set is significantly simplified by making the database size divisible by this factor of 2. Since it makes less work for us in the long run, and does not impede the general applicability of the sizing scheme to other multi-backend database systems, we include the requirement here with the general database-size considerations.

Therefore, we have (N/16)/2, which means that N must be a multiple of 32 times the LCM{1,2,....,M} times the record size.

TABLE 27. TEST CONFIGURATIONS FOR LARGE DATABASE.

| Configuration Number | Number of Backends | Record Size in Bytes | Number of Records per Backend | Mbytes per Backend | Database Size in Mbytes |
|---|---|---|---|---|---|
| 1 | 1 | 2000 | 49,920 | 99.84 | |
| | | 1000 | 99,840 | 99.84 | |
| | | 400 | 249,600 | 99.84 | 399.36 |
| | | 200 | 499,200 | 99.84 | |
| 2 | 2 | 2000 | 24,960 | 49.92 | |
| | | 1000 | 49,920 | 49.92 | |
| | | 400 | 124,800 | 49.92 | 399.36 |
| | | 200 | 249,600 | 49.92 | |
| 3 | 3 | 2000 | 16,640 | 33.28 | |
| | | 1000 | 33,280 | 33.28 | |
| | | 400 | 83,200 | 33.28 | 399.36 |
| | | 200 | 166,400 | 33.28 | |
| 4 | 4 | 2000 | 12,480 | 24.96 | |
| | | 1000 | 24,960 | 24.96 | |
| | | 400 | 62,400 | 24.96 | 399.36 |
| | | 200 | 124,800 | 24.96 | |
| 5 | 2 | 2000 | 49,920 | 99.84 | |
| | | 1000 | 99,840 | 99.84 | |
| | | 400 | 249,600 | 99.84 | 798.72 |
| | | 200 | 499,200 | 99.84 | |
| 6 | 3 | 2000 | 49,920 | 99.84 | |
| | | 1000 | 99,840 | 99.84 | |
| | | 400 | 249,600 | 99.84 | 1,198.08 |
| | | 200 | 499,200 | 99.84 | |
| 7 | 4 | 2000 | 49,920 | 99.84 | |
| | | 1000 | 99,840 | 99.84 | |
| | | 400 | 249,600 | 99.84 | 1,597.44 |
| | | 200 | 499,200 | 99.84 | |

Let us summarize these database design considerations. First, we decide to test with three database sizes, (small = N/4, medium = N/2, and large = N). The largest database is approximately the maximum formated capacity of a backend's secondary storage system, (which is obviously hardware dependent).

Second, to determine the largest feasible upper-bound for each test database set, we find the corresponding multiple of database size from Table 2. If the system being evaluated has three backends, N must be divisible by (6 x 32 x rec-size); if the system has four backends, N must be divisible by (12 x 32 x rec-size); etc. The (X x 32) portion of the database-size multiple, not including rec-size, is used to determine an upper-bound for the large database size, N. In the example

we considered, we found that this upper-bound was N = 399.999744 Mbytes, since N had to be divisible by 12 x 32 = 384 for a 4 backend system.

Third, we consider the record-size parameter, which is hardware dependent. We select four record sizes based on the size of the unit of data storage and access used by the particular system's architecture. We select one large and one small size, with two intermediate values. We require that the largest record size be divisible by each of the three smaller record sizes to simplify the database sizing process.

Fourth, we calculate the required database multiple in accordance with Table 2, using the **largest** record size selected above in step 3 for the rec-size parameter value. Since the other three record sizes are divisors of the rec-size parameter, we are assured that the database we create using this database multiple will be divisible by all four record types.

Fifth, since the database multiple is now known, we calculate the required values of N, N/2, and N/4, respectively. Substitution of N, N/2, and N/4 for N in the applicable system configuration table will enable us to verify that these three databases are feasible, since they permit each database to be split evenly as required for all of the necessary test configurations.

Sixth, we decide how to format the actual test databases. Two options seem feasible: (a) use only one record type per database; (b) include all four record types in a single database. If the system being tested can accommodate multiple record types within a database, then option (b) seems to be the best choice. With this option for our four-backend example, less work is involved, since only three test databases need to be created. If we select option (a) instead, then twelve test databases would be required for our example. We recommend option (b) to streamline the performance evaluation task. For the hypothetical system we considered, we derived system configuration Tables 12-23 for case (a), and Tables 25-27 for case (b). These tables show that whichever option is selected, our design results in a test database set in which each database may be evenly split as required for all of the necessary test configurations.

## C. SYSTEM-DEPENDENT FACTORS

At this point, we leave the discussion of database design considerations. We have presented a methodology for designing a test database set, including selection of record sizes. and have shown that this design satisfies the requirements for accommodating all required test system configurations. Beyond this point, database design factors tend to become system specific.

First, the data model used by the database system being evaluated must be considered, since it is directly related to the system's data management strategy. including such factors as the directory structure and record distribution mechanism used by the system.

Similarly, record composition, (i.e., the makeup of record fields), may rely on specific system idioms and constraints, such as limits on field width, or on the number of fields within a record. etc. Such considerations therefore impede the design and development of a generalized test database set. These factors will be considered in Chapter V when we design a specific database set for use in evaluating the Multi-Backend Database System. MBDS.

## D. TEST-TRANSACTION MIX CONSIDERATIONS

As noted earlier. if we are to demonstrate the response-time invariance of software multi-backend database systems, we must ensure that any increase in the number of backends in the system is accompanied by a proportional increase in the size of the database. and in the size of the response set returned by the test-transaction mix. Table 4 cites the obvious size parameter required for these system tests.

However, the selection of a test-transaction mix which will permit the database size to increase in the same proportion as the increase in the response set size is much more complex. The selection requires a complete understanding of the characteristics and features of the data model and data manipulation language. Also, the directory structure and storage strategies of the system play a major role. Using the Naval Postgraduate School's MBDS. we will show how to cleverly design a test-record organization, a test-database structure, and a test-transaction mix set which enables the system evaluator to use the same

organization, structure, and mix for all system configurations **without modification!**

By careful consideration of the specification of the directory structure, and construction of the test-transaction mix, the requests that we select will ensure us that the database size will increase in exactly the same proportion as the increase in the response-set size. This feature greatly simplifies the actual test execution process, and strengthens our test reliability and validity claims.

A second goal of the thesis is to develop a test-transaction mix to test the overall performance of MBDS. Hawthorn and Stonebraker, [Ref. 13: pp. 3-4], suggest that three sets of test transactions be used to support this task. One set consists of **overhead-intensive queries** for which the actual time required to process the required data is much less than the system overhead required to process the request. The **data processing time** is defined as the time required for the DBMS to fetch and manipulate the required data, whereas **system overhead** involves both operating system and DBMS time involved with such tasks as user communication, and query parsing and validity checking. In essence, overhead-intensive queries reference very little data [Ref. 13: p. 3].

The second type of transaction is called a **data-intensive query**. In this type of transaction, the data processing time is much greater than the system overhead. Therefore, data-intensive queries reference large quantities of data [Ref. 13: pp. 3-4]. Finally, the last type of transaction, called **multi-relation queries**, are geared to relational systems for transactions which involve more than one relation. Therefore, they involve a relational **join** operation, similar to the MBDS operation **retrieve-common** [Ref. 13: p. 4].

We will consider all of these factors when selecting transactions to verify the performance-gain and capacity-growth claims, and to measure overall MBDS system performance.

# IV. THE MULTI-BACKEND DATABASE SYSTEM (MBDS)

Figure 8 shows the basic MBDS architectural organization. One microcomputer functions as the controller, while one or more microcomputers and their disk systems serve as backends. Both the controller and the backends are interconnected by a broadcast bus. Together, the controller, the broadcast bus, and the backends comprise a database system which is specifically designed to overcome the performance and capacity-growth problems experienced by traditional mainframe-based and conventional software single-backend database systems. The initial design and analysis of MBDS is presented in [Ref. 7 and Ref. 14], while the implementation efforts are documented in [Ref. 9, Ref. 15, Ref. 16, and Ref. 17].

In this chapter we present a brief overview of the MBDS. First, we discuss the system architecture, and describe the prototype configuration, as well as the interim-system upgrade. Then, we describe the attribute-based data-model, the directory tables, and the attribute-based data-language (ABDL). Finally, we discuss the directory and database placement, and the MBDS process structure. The material presented in this chapter is primarily extracted from [Ref. 3: pp. 10-27].

## A. THE MBDS ARCHITECTURAL ORGANIZATION

As an interim system, the prototype MBDS configuration has a VAX-11/780 (VMS OS) minicomputer as the controller and two PDP-11/44 (RSX-11M OS) minicomputers and their disk systems as the backends. Each backend uses a single DEC RM02 disk drive with a maximum formated capacity of 67-Mbytes, a peak transfer rate of 806-Kbytes per second, and an average access time of 42.5 ms (30 ms average seek time + 12.5 ms average latency time). Intercomputer communication is performed by time-division-multiplexed buses which are known as parallel communications links (PCL-11Bs). [Ref. 3: p. 27].

MBDS is a message-oriented system [Ref. 17]. Consequently, each system process corresponds to a unique system function. The MBDS processes

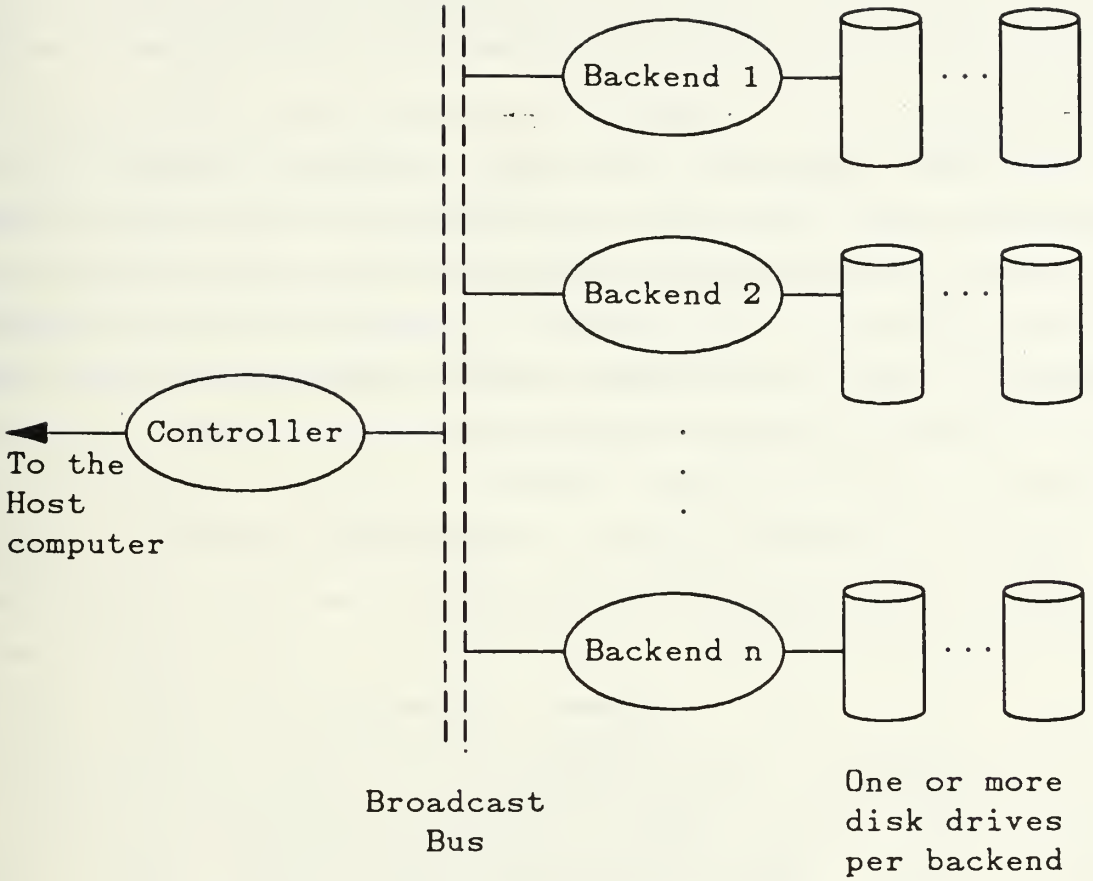


Figure 8. The MBDS Architectural Organization.

communicate by passing messages. When it receives a user transaction, the controller broadcasts the transaction to each backend. Since broadcast-based communications devices, such as the Ethernet, were not available in 1980 when MBDS development began, a software interface was provided for each computer in the MBDS prototype system to emulate the broadcast capability using the PCL. Each MBDS computer, whether it is the controller or a backend, has two complimentary communications processes. The **get-pcl** process receives messages from other computers via the PCL, while the **put-pcl** process puts messages onto the PCL bus to be broadcast to other computers in the system [Ref. 3: pp. 10-11].

The data in the MBDS system is distributed across the disk systems of each backend computer. Consequently, a user transaction may be executed simultaneously by all backends. Each backend maintains a transaction queue, and schedules the processing of each transaction independent of the other

backends. This enables each backend to maximize its access operations, and to minimize its idle time. Therefore, the backends can process requests in parallel [Ref. 3: p. 10].

The MBDS designers took great care to minimize the potential for the controller to become a bottleneck. To minimize the work done by the controller, the designers made the backends perform all of the primary database record processing functions. The controller is restricted to less demanding administrative tasks concerned with transaction preparation and broadcasting, post processing and routing of results back to the host, and directing the insertion of new data records [Ref. 3: p. 10].

The prototype MBDS configuration enabled the system developers to proceed with design and implementation work long before broadcast-bus technology and 32-bit-microprocessors became available. However, the use of PDP-11/44s as backends created a number of problems. With the PDP-11/44, each backend is limited to 256-Kbytes of physical memory, and to 64-Kbytes of virtual memory space per process. Since the MBDS functions are implemented as processes, we have large processes. These restrictions forced the developers to design overlays to enable them to fit each backend process into the restricted virtual memory space. In addition, testing was limited to a maximum record size of 200-bytes per record, with a maximum of 1000 records in the test database. Since MBDS was specifically designed as a high-performance system for very large databases, these restrictions severely limited the amount and type of testing which could be accommodated. The need to simulate the broadcast capability with PCLs further complicated the software structure, and also added to system overhead [Ref. 3: pp. 27-28].

We are presently working on an MBDS hardware upgrade featuring ten Sun-2/170 workstations (4.2 BSD Unix OS). One Sun workstation will serve as the controller, while the other nine will function as backends. The Sun-2/170 workstation uses the Motorola MC68010 32-bit microprocessor as the CPU, and features 16-Mbytes of virtual memory space per process. The new MBDS configuration will use Ethernet as the broadcast bus among workstations. Initially, each backend will have one dedicated Fujitsu Eagle Winchester-type

disk drive with a maximum formated capacity of 380-Mbytes per drive [Ref. 3: p. 28].

The new MBDS system configuration will eliminate the restrictions inherent in the VAX/PDP prototype version. The use of a broadcast-based communications network will eliminate the overhead experienced with the get-pcl/put-pcl software broadcast emulation. Furthermore, the larger virtual memory will remove the record-size and database-size restrictions imposed by the PDP-11/44 architecture. When the hardware installation and software conversion to the new Sun/Unix environment is completed, we will conduct a complete performance evaluation of MBDS. This thesis, therefore, provides a stepping-stone to this performance evaluation effort by presenting the methodology to be used for the evaluation of the performance-gain and capacity-growth goals of MBDS.

## B. THE ATTRIBUTE-BASED DATA MODEL

The MBDS is based on the attribute-based data model, which was first proposed in [Ref. 18]. The material included in this section to describe the model is extracted from [Ref. 3: pp. 11-12].

In the attribute-based data model, data is considered in terms of the following constructs: database, file, record, attribute-value pair, keyword, attribute-value range, directory keyword, non-directory keyword, directory, record body, keyword predicate, and query. We define **database** as a collection of files. Each **file** contains a group of records which are characterized by a unique set of directory keywords. A **record** has two major components. First, we have a collection of **attribute-value pairs** or **keywords**. Each attribute-value pair of a record is a member of the set formed by taking the cartesian product of the attribute name and its value domain. For example, in the attribute-value pair <POPULATION, 25000>, the population attribute has a value of 25000. Each record may contain a maximum of one attribute-value pair for each attribute defined for the database. For the **directory keywords** of a record (or a file), their attributes, known as directory attributes, are kept in a **directory** and are used for identifying the records (files). All attribute-value pairs whose attributes are not kept in the directory are called **non-directory**

**keywords**. The second record component is the **record body**, which consists of miscellaneous textual information. Figure 9 depicts a sample record. Note in Figure 9 that the record is enclosed in parentheses. Angle brackets, <,>, enclose the keyword attribute-value pairs, while curly brackets, {,}, enclose the record body. To identify the specific file being referenced, the first attribute-value pair of all records within a file is the same. Specifically, the first attribute of each record, denoted as FILE, is associated with the corresponding file name value. Therefore, the sample record in Figure 9 is from the USCensus file.

```
(<FILE, USCensus>, <CITY, Monterey>, <POPULATION, 25000>, {Temperate climate})
```

Figure 9. Sample Record Format.

The user may identify database records by keyword predicates. A **keyword predicate** is a tuple which consists of a directory attribute, a relational operator ($=$, $\neq$, $>$, $<$, $\geq$, $\leq$), and an attribute value. For example, POPULATION $\geq$ 15000 is a greater-than-or-equal-to keyword predicate. A database **query** is formed by combining keyword predicates in disjunctive normal form. The sample query shown in Figure 10 will be satisfied by all records of the USCensus file with a CITY value of either Monterey or San Jose. (Note that we use parentheses for bracketing conjunctions within a query to improve clarity.)

```
(FILE = USCensus and CITY = Monterey) or (FILE = USCensus and CITY = San Jose)
```

Figure 10. Sample Database Query.

## C. THE MBDS DIRECTORY TABLES

MBDS uses a set of directory tables to manage the database. The directory data is described via the following constructs: attributes, descriptors, and clusters. Each **attribute** represents a category of the user data. For example,

the POPULATION attribute corresponds to actual populations stored in the database. A **descriptor** describes either the range of values or the exact value that a directory attribute can have. Therefore, (50,001 ≤ POPULATION ≤ 100.000) is a possible descriptor for the POPULATION attribute. The descriptors that we define for an attribute (such as population ranges) are mutually exclusive.

Finally, we define a **cluster** as a group of records for which every record in the cluster satisfies the same set of descriptors. Therefore, all of the records with POPULATION between 50,001 and 100,000 may form a cluster for the descriptor cited above. In this example, the cluster satisfies the set of a single descriptor. In general, a cluster will satisfy one or more descriptors, known as the **descriptor set**.

There are three basic tables in the MBDS directory structure. The **attribute table** (AT) maps the directory keyword attributes to their corresponding descriptors. Table 28 depicts a sample AT. The **descriptor-to-descriptor-id table** (DDIT) maps each descriptor to a unique descriptor identifier. A sample DDIT is shown in Table 29. Finally, the **cluster-definition table** (CDT) maps descriptor-id sets to cluster ids. Each entry of the CDT consists of a unique cluster-id, the set of descriptor-ids for the descriptors which define the cluster, and the corresponding disk address of each record in the clusters. Table 30 shows a sample CDT. To access records in the database, MBDS must first access the directory data contained in the AT, DDIT, and CDT tables.

TABLE 28. AN ATTRIBUTE TABLE (AT).

| Attribute | Pointer |
|---|---|
| POPULATION | P |
| CITY | C |
| FILE | F |

TABLE 29. A DESCRIPTOR-TO-DESCRIPTOR-ID-TABLE (DDIT).

| Descriptor | Dij |
|---|---|
| 0 ⩽ POPULATION ⩽ 50,000 | D11 |
| 50,001 ⩽ POPULATION ⩽ 100,000 | D12 |
| 100,001 ⩽ POPULATION ⩽ 250,000 | D13 |
| 250,001 ⩽ POPULATION ⩽ 1,000,000 | D14 |
| CITY = Cumberland | D21 |
| CITY = Columbus | D22 |
| CITY = Monterey | D23 |
| CITY = Toronto | D24 |
| FILE = CanadaCensus | D31 |
| FILE = USCensus | D32 |

Dij: Descriptor j for attribute i.

TABLE 30. A CLUSTER-DEFINITION TABLE (CDT).

| Id | Descriptor-Id Set | Record-Id |
|---|---|---|
| C1 | {D11,D21,D32} | A1,A2 |
| C2 | {D14,D22,D32} | A3 |
| C3 | {D12,D23,D32} | A4 |
| C4 | {D14,D24,D31} | A5 |

MBDS uses three types of descriptors. A **type-A** descriptor is a conjunction of a less-than-or-equal-to predicate and a greater-than-or-equal-to predicate, such that the same attribute appears in both predicates. For example, ((POPULATION ⩾ 10000) and (POPULATION ⩽ 15000)) is a type-A descriptor. A **type-B** descriptor consists of only an equality predicate. (FILE = USCensus) is an example of a type-B descriptor. Finally, a **type-C** descriptor consists of the name of an **attribute**. The type-C attribute defines a set of type-C sub-descriptors. **Type-C sub-descriptors** are equality predicates defined over all unique attribute values which exist in the database. For example, the type-C attribute CITY forms the type-C sub-descriptors (CITY=Cumberland), (CITY=Columbus), (CITY=Monterey) and (CITY=Toronto), where

"Cumberland", "Columbus", "Monterey", and "Toronto" are the only unique database values for the CITY attribute [Ref. 3: pp. 12-14].

## D. THE ATTRIBUTE-BASED DATA LANGUAGE (ABDL)

The attribute-based data language (ABDL) [Ref. 7: pp. 67-77, and Ref. 15: pp. 43-49] serves as the MBDS data manipulation language. In this section we provide a brief overview of the five primary database operations, INSERT, DELETE, UPDATE, RETRIEVE, and RETRIEVE-COMMON. See [Ref. 9: pp. 55-75, and Ref. 19] for more detailed descriptions of request execution in the MBDS. The material in this section is extracted from [Ref. 3: pp. 14-16].

Each ABDL **request** consists of a primary database operation with a qualification. The **qualification** part is used to specify the database records that are to be operated on. A **transaction** consists of two or more requests which are grouped together. In this section we will illustrate each of the five types of requests.

New records are inserted into the database via the **INSERT** request. The qualification of an INSERT request consists of a list of keywords and a record body. Figure 11 shows an INSERT request to insert a record into the USCensus file for the city Cumberland with a population of 40,000.

```
INSERT (<FILE, USCensus>, <CITY, Cumberland>, <POPULATION, 40,000>)
```

Figure 11. Sample INSERT Request.

We remove record(s) from the database via the **DELETE** request. The qualification of a DELETE request is a query. Figure 12 depicts a request to delete all records from the USCensus file whose population is greater than 100,000.

We may modify records in the database via the **UPDATE** request. An UPDATE request qualification consists of two parts, the query and the modifier. The **query** specifies which database records are to be modified, while the **modifier** specifies how the records being modified are to be updated. Figure 13

```
DELETE ((FILE = USCensus) and (POPULATION > 100,000))
```

Figure 12. Sample DELETE Request.

depicts an UPDATE request to modify all records of the USCensus file by increasing all populations by 5,000. In this example, ((FILE = USCensus)) is the query, and (POPULATION = POPULATION + 5,000) is the modifier.

```
UPDATE (FILE = USCensus) (POPULATION = POPULATION + 5,000)
```

Figure 13. Sample UPDATE Request.

We retrieve records from the database via the **RETRIEVE** request. The retrieve request qualification consists of a query, a target-list, and a by-clause. The **query** specifies which records are to be retrieved. The **target-list** consists of a list of attributes to be output. It may also consist of an **aggregate operation**, (AVG, COUNT, SUM, MIN, MAX), on one or more of the output attributes. When an aggregate operation is specified, an optional **by-clause** may be used to group records. The RETRIEVE request shown in Figure 14 will retrieve the city names of all records in the USCensus file with populations greater than or equal to 50,000. The query portion is denoted by ((FILE = USCensus) and (POPULATION $\geq$ 50,000)), while the target-list consists of the CITY attribute. This example does not use a by-clause or an aggregate operation.

```
RETRIEVE ((FILE = USCensus) and (POPULATION ≥ 50,000)) (CITY)
```

Figure 14. Sample RETRIEVE Request.

The **RETRIEVE-COMMON** request is used to merge two files by common attribute-values. Logically, the RETRIEVE-COMMON request can be considered as two retrieve requests that are processed serially in the general form shown in Figure 15.

```
RETRIEVE (query-1) (target-list-1)
COMMON   (attribute-1, attribute-2)
RETRIEVE (query-2) (target-list-2)
```

Figure 15. Format of the RETRIEVE-COMMON Request.

Attribute-1 (associated with the first retrieve request) and attribute-2 (associated with the second retrieve request) are the common attributes. Figure 16 depicts a RETRIEVE-COMMON request which will find all records in both the CanadaCensus file and the USCensus file for which population is greater than 100,000. Then, it identifies the records from this common set of records which have equal population figures, and returns the city name(s) for those records which have identical population values.

```
RETRIEVE ((FILE = CanadaCensus) and (POPULATION ≥ 100,000)) (CITY)
COMMON (POPULATION, POPULATION)
RETRIEVE ((FILE = USCensus) and (POPULATION ≥ 100,000)) (CITY)
```

Figure 16. Sample RETRIEVE-COMMON Request.

## E. THE DIRECTORY AND DATABASE PLACEMENT

The directory tables and user data records are stored on the secondary memory devices of the backends. The directory tables are replicated at each backend, while the user data records are distributed evenly across all of the backends. The material in this section is extracted from [Ref. 3: pp. 16-17].

Each backend maintains its own copy of the directory tables. These directory tables are identical, except for the record-id field of the CDT. Since the record ids of each CDT represent the secondary-storage addresses of the records stored on the specific backend's disks, they are unique for each backend. The directory tables are stored in the secondary memory, and are staged into the primary memory when required for processing. Usually ten to twenty percent of the size of the user data, i.e., of the entire database size, is reserved to accommodate the directory tables in secondary storage [Ref. 3: p. 16].

The MBDS uses a cluster-based database placement algorithm to distribute records across the backends. When a new record is to be inserted, the controller selects the backend to receive the new record. The chosen backend will insert the new record into a block of its secondary storage. A backend may continue to insert additional new records for the same cluster into the block until the block is filled. When this occurs, the backend sends the controller a "block-is-full" message. The controller then selects another backend to continue with the insertion of new records for the same cluster. The controller maintains a list of those backends which have room in their secondary-storage blocks for the insertion of new records into the existing clusters [Ref. 3: p. 17]. We will consider database placement again in Chapter V when we design the test-database files.

## F. THE PROCESS STRUCTURE

In addition to the get-pcl and put-pcl communications processes which we discussed earlier, there are seven other MBDS processes which are created at system start-up, and which exist until the system is stopped. We will briefly describe each of these processes in this section. Figure 17 depicts an overview of the MBDS process structure.

Although the MBDS controller is intended to interface with a host computer, MBDS may also interface with a user terminal. The **test-interface** process which was developed by Kovalchik [Ref. 5] provides a means for the user to interface with MBDS via a terminal. The menu-driven test-interface process executes on the controller, and enables the user to create a new database, load test files and create required directory entries, and create, execute, and/or archive database test transactions [Ref. 3: p. 18].

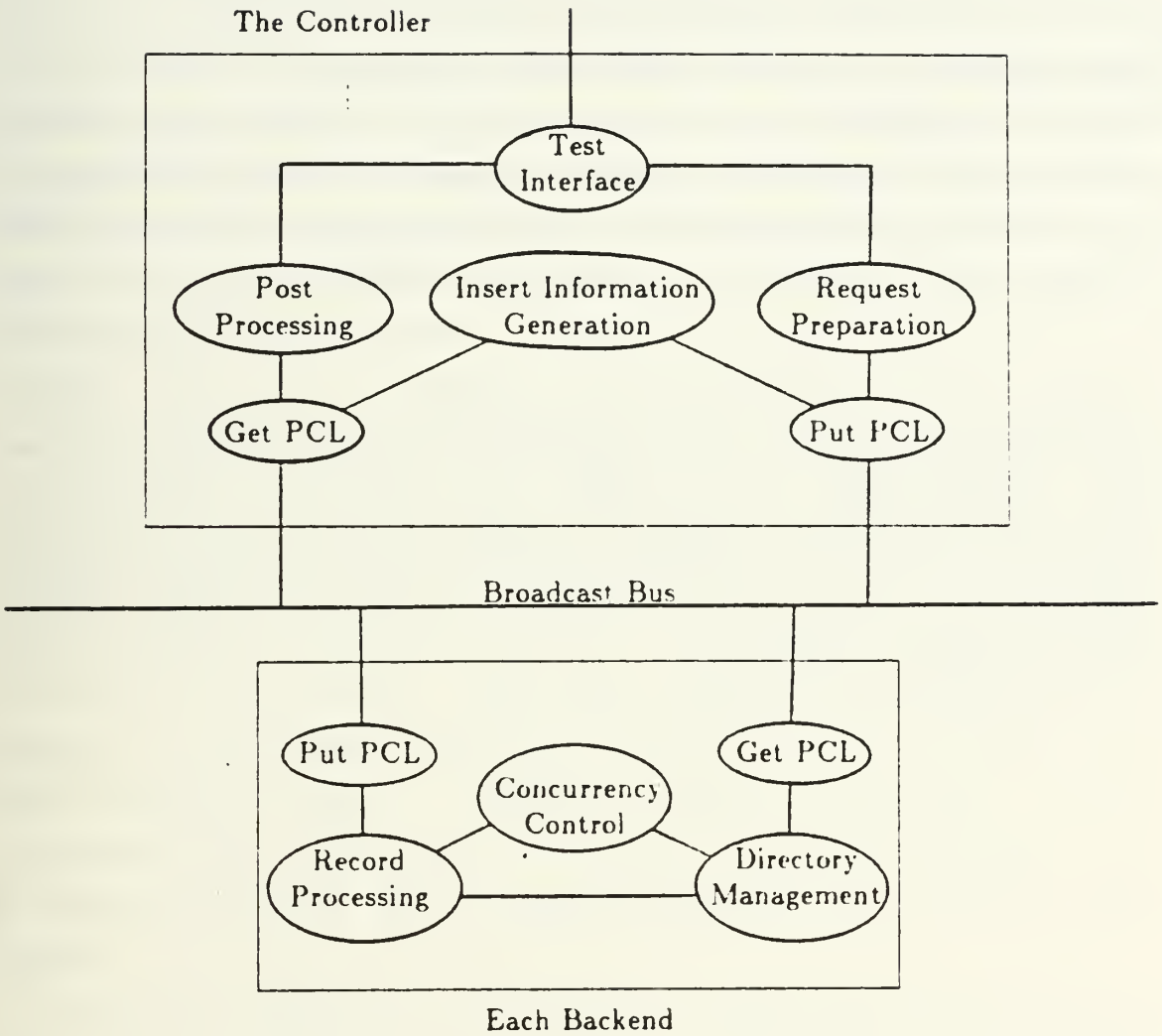


Figure 17. The MBDS Process Structure.

Besides the get-pcl, put-pcl, and test-interface processes, the controller includes the request preparation, insert information generation, and post processing processes. **Request preparation** is responsible for receiving, parsing, and formating requests before sending them to the directory-management process in each backend. The **insert information generation** process in the controller determines the backend at which to insert a new record. Finally, **post**

**processing** collects the results of a request (transaction), and forwards the results to the user when processing is complete [Ref. 3: p. 19].

Each backend consists of the get-pcl and put-pcl communication processes, and three other processes named directory management, concurrency control, and record processing. Essentially, **directory management** controls request execution at the backend, and manages the secondary-memory-based directory tables (AT, DDIT, and CDT). Directory management determines the relevant record disk addresses from the CDT. It then passes these addresses to record processing. The **concurrency control** process arbitrates the access to the directory data and user data records to ensure database and directory consistency, while permitting concurrent execution of user requests. Finally, **record processing** performs the actual database operation specified by the user request, including disk I/O operations. It receives the secondary-storage addresses from directory management, (as noted above). When processing is complete, record processing forwards the results to the post-processing process in the controller [Ref. 3: pp. 19-20].

# V. THE DESIGN OF THE MBDS TEST-DATABASE SET

In this chapter we consider database design factors for the multi-backend database system (MBDS). In the first section we size the MBDS test-database set, and develop the database format based on the MBDS storage mechanism, which stores records into record clusters. In the second section, we design record templates for the four desired record classes, and define descriptors for the corresponding cluster categories.

## A. THE DATABASE-SIZE COMPUTATION AND CLUSTER FORMATION

As discussed in Chapter IV, the MBDS hardware is being upgraded. The new system will use Sun-2/170 workstations, where each backend system utilizes at least a single Fujitsu Eagle Winchester-type disk drive with a maximum formated capacity of 380 Mbytes per drive. Each backend will have one drive dedicated for the test database. Since MBDS duplicates the directory data at all backends, we will reserve 80 Mbytes of the total disk capacity of each disk for the MBDS directory [Ref. 16: p. 7]. Therefore, the MBDS test-database set will have a restriction of 300 Mbytes in size. In summary, the reserved **directory size** is 80 Mbytes per backend, and the reserved **database size** is 300 Mbytes per backend.

Our second consideration is **record size**. As discussed in Chapter III, we base record-size selection on the track size to be supported by the Sun/Unix environment. Since we expect the new Fujitsu disks to use a 16-Kbyte track size. the block size we select must divide evenly into the 16-Kbyte track to permit us to store a whole number of blocks in each track. Therefore, for the MBDS testing scheme to be described in this section, we select a 4-Kbyte block size, and select four record sizes of 2000, 1000, 400, and 200 bytes per record, as described in Chapter III.

The next step is to calculate the **database multiple** (DBM). As described in Chapter III, this calculation is based on the relationship:

$$DBM = (LCM\{1,2,.....M\} \times 32 \times \text{rec-size}).$$

where M is the maximum number of backends to be used for testing. In the material which follows, we use a sample system with a maximum of three backends to illustrate the steps involved in designing a test-database set for the MBDS system.

For an MBDS with three backends, the corresponding database multiple is:

$$\begin{aligned} DBM &= (LCM\{1,2,3\} \times 32 \times \text{rec-size}) \\ &= 6 \times 32 \times 2000 \\ &= 384{,}000. \end{aligned}$$

Therefore, the size of N in Mbytes of the largest database is the largest multiple of 384,000 bytes which is less than or equal to 300 Mbytes. Consequently, we calculate the following:

| | | | | |
|---|---|---|---|---|
| N | = | 299.904 Mbytes | --> | {Large database} |
| N/2 | = | 149.952 Mbytes | --> | {Medium database} |
| N/4 | = | 74.976 Mbytes | --> | {Small database}. |

Table 31 lists MBDS test-database set sizes for from two to eleven backends, assuming an upper-bound size restriction of 300 Mbytes of database storage per backend. Note that for the specific MBDS configuration that we are considering, the test scheme will only suffice for up to ten backends! The three database sizes listed for the eleventh backend all exceed the 300 Mbyte size restriction! Therefore, to test MBDS with eleven or more backends, we must have more than one disk drive per backend. If we assume a relationship of 300 Mbytes for the database and 80 Mbytes for directories per disk, then we would need six disks connected to a single backend to accommodate a database of size N = 1,774.08 Mbytes! System evaluators and developers must consider these factors when proposing future system expansion and performance evaluation.

Table 32, (which we originally presented as Table 6 in Chapter III), lists the corresponding test configurations required for an MBDS with three backends. Since the MBDS version to be tested will support multiple record templates within a database, we elect to include all four record types in a single database. This will require a test-database set consisting of three databases representing the small, medium, and large categories. Adhering to the methodology of Chapter III, we derive Tables 33 to 35 for the MBDS test-database set for small, medium. and large databases consisting of four record groupings per database, with records of 2000, 1000, 400, and 200 bytes-per-record.

TABLE 31. MBDS DATABASE SIZE CALCULATIONS.

| M | LCM of {1,...,M} | DBM in Bytes | N in Mbytes | N/2 in Mbytes | N/4 in Mbytes |
|---|---|---|---|---|---|
| 2 | 2 | 128,000 | 299.904 | 149.952 | 74.976 |
| 3 | 6 | 384,000 | 299.904 | 149.952 | 74.976 |
| 4 | 12 | 768,000 | 299.520 | 149.760 | 74.880 |
| 5 | 60 | 3,840,000 | 299.520 | 149.760 | 74.880 |
| 6 | 60 | 3,840,000 | 299.520 | 149.760 | 74.880 |
| 7 | 420 | 26,880,000 | 295.680 | 147.840 | 73.920 |
| 8 | 840 | 53,760,000 | 268.800 | 134.400 | 67.200 |
| 9 | 2,520 | 161,280,000 | 161.280 | 80.640 | 40.320 |
| 10 | 2,520 | 161,280,000 | 161.280 | 80.640 | 40.320 |
| 11 | 27,720 | 1,774,080,000 | 1,774.080 | 887.040 | 443.520 |

where:
M = maximum number of backends in the database.
LCM = Least Common Multiple.
DBM = (LCM{1,...,M} * 32 * rec-size) for rec-size = 2000-bytes.
N = Size in Mbytes of large test database.
N/2 = Size in Mbytes of medium size test database.
N/4 = Size in Mbytes of small test database.

Assumption: Largest database allowable is 300 Mbytes.

TABLE 32. TEST CONFIGURATIONS WITH THREE BACKENDS.

| Configuration Number: | Number of Backends: | Mbytes per Backend: | Total database Size in Mbytes |
|---|---|---|---|
| 1 | 1 | N | N |
| 2 | 2 | N/2 | N |
| 3 | 3 | N/3 | N |
| 4 | 2 | N | 2N |
| 5 | 3 | N | 3N |

Note:
Configuration's {1,2,3} are required to verify goal (1).
Configuration's {1,4,5} are required to verify goal (2).

TABLE 33. SMALL DATABASE TEST CONFIGURATIONS.

| Configuration Number | Number of Backends | Record Size in Bytes | Number of Records per Backend | Mbytes per Backend | Database Size in Mbytes |
|---|---|---|---|---|---|
| 1 | 1 | 2000 | 9,372 | 18.744 | |
| | | 1000 | 18,744 | 18.744 | |
| | | 400 | 46,860 | 18.744 | 74.976 |
| | | 200 | 93,720 | 18.744 | |
| 2 | 2 | 2000 | 4,686 | 9.372 | |
| | | 1000 | 9,372 | 9.372 | |
| | | 400 | 23,430 | 9.372 | 74.976 |
| | | 200 | 46,860 | 9.372 | |
| 3 | 3 | 2000 | 3,124 | 6.248 | |
| | | 1000 | 6,248 | 6.248 | |
| | | 400 | 15,620 | 6.248 | 74.976 |
| | | 200 | 31,240 | 6.248 | |
| 4 | 2 | 2000 | 9,372 | 18.744 | |
| | | 1000 | 18,744 | 18.744 | |
| | | 400 | 46,860 | 18.744 | 149.952 |
| | | 200 | 93,720 | 18.744 | |
| 5 | 3 | 2000 | 9,372 | 18.744 | |
| | | 1000 | 18,744 | 18.744 | |
| | | 400 | 46,860 | 18.744 | 224.928 |
| | | 200 | 93,720 | 18.744 | |

TABLE 34. MEDIUM DATABASE TEST CONFIGURATIONS.

| Configuration Number | Number of Backends | Record Size in Bytes | Number of Records per Backend | Mbytes per Backend | Database Size in Mbytes |
|---|---|---|---|---|---|
| 1 | 1 | 2000 | 18,744 | 37.488 | |
| | | 1000 | 37,488 | 37.488 | |
| | | 400 | 93,720 | 37.488 | 149.952 |
| | | 200 | 187,440 | 37.488 | |
| 2 | 2 | 2000 | 9,372 | 18.744 | |
| | | 1000 | 18,744 | 18.744 | |
| | | 400 | 46,860 | 18.744 | 149.952 |
| | | 200 | 93,720 | 18.744 | |
| 3 | 3 | 2000 | 6,248 | 12.496 | |
| | | 1000 | 12,496 | 12.496 | |
| | | 400 | 31,240 | 12.496 | 149.952 |
| | | 200 | 62,480 | 12.496 | |
| 4 | 2 | 2000 | 18,744 | 37.488 | |
| | | 1000 | 37,488 | 37.488 | |
| | | 400 | 93,720 | 37.488 | 299.904 |
| | | 200 | 187,440 | 37.488 | |
| 5 | 3 | 2000 | 18,744 | 37.488 | |
| | | 1000 | 37,488 | 37.488 | |
| | | 400 | 93,720 | 37.488 | 449.856 |
| | | 200 | 187,440 | 37.488 | |

TABLE 35. LARGE DATABASE TEST CONFIGURATIONS.

| Configuration Number | Number of Backends | Record Size in Bytes | Number of Records per Backend | Mbytes per Backend | Database Size in Mbytes |
|---|---|---|---|---|---|
| 1 | 1 | 2000 | 37,488 | 74.976 | |
| | | 1000 | 74,976 | 74.976 | |
| | | 400 | 187,440 | 74.976 | 299.904 |
| | | 200 | 374,880 | 74.976 | |
| 2 | 2 | 2000 | 18,744 | 37.488 | |
| | | 1000 | 37,488 | 37.488 | |
| | | 400 | 93,720 | 37.488 | 299.904 |
| | | 200 | 187,440 | 37.488 | |
| 3 | 3 | 2000 | 12,496 | 24.992 | |
| | | 1000 | 24,992 | 24.992 | |
| | | 400 | 62,480 | 24.992 | 299.904 |
| | | 200 | 124,960 | 24.992 | |
| 4 | 2 | 2000 | 37,488 | 74.976 | |
| | | 1000 | 74,976 | 74.976 | |
| | | 400 | 187,440 | 74.976 | 599.808 |
| | | 200 | 374,880 | 74.976 | |
| 5 | 3 | 2000 | 37,488 | 74.976 | |
| | | 1000 | 74,976 | 74.976 | |
| | | 400 | 187,440 | 74.976 | 899.712 |
| | | 200 | 374,880 | 74.976 | |

As discussed above, we have decided to use record sizes of 2000, 1000, 400, and 200 bytes per record, based on the fact that the MBDS will process information from the secondary memory via a 4-Kbyte block. These record sizes produce a range of from 2 to 20 records per block, as shown in Table 36. Given that we have four record classes in our test database, we must now determine how to distribute these records within the database. Therefore, we must consider the MBDS storage mechanism.

TABLE 36. THE RECORDS-PER-BLOCK RELATIONSHIP.

| Record Size in Bytes | Records per Block |
|---|---|
| 2000 | 2 |
| 1000 | 4 |
| 400 | 10 |
| 200 | 20 |

Recall from Chapter IV that MBDS stores records in clusters. We have selected nine cluster categories, with each cluster containing from 2 to 10 blocks of records per cluster. This design provides a uniform range of cluster sizes which facilitates the design of an extensible and versatile test-transaction mix. Table 37 lists the number of records per cluster for each of the four record types. Note that Table 37 makes use of the records-per-block relationship of Table 36 for each record type. For example, the cluster category with two blocks per cluster has four 2000-byte records per cluster, eight 1000-byte records per cluster, twenty 400-byte records per cluster, and forty 200-byte records per cluster. These values are calculated by multiplying the number of records per block by the number of blocks per cluster, (i.e., the records-per-block column of Table 36 by the blocks-per-cluster column of Table 37).

TABLE 37. NUMBER OF RECORDS PER CLUSTER CATEGORY.

| Blocks per Cluster: | Record Size in Bytes: | | | |
|---|---|---|---|---|
| | 2000 | 1000 | 400 | 200 |
| 2 | 4 | 8 | 20 | 40 |
| 3 | 6 | 12 | 30 | 60 |
| 4 | 8 | 16 | 40 | 80 |
| 5 | 10 | 20 | 50 | 100 |
| 6 | 12 | 24 | 60 | 120 |
| 7 | 14 | 28 | 70 | 140 |
| 8 | 16 | 32 | 80 | 160 |
| 9 | 18 | 36 | 90 | 180 |
| 10 | 20 | 40 | 100 | 200 |

Our last consideration is to determine how many clusters of each cluster category should be chosen for each of the four record classes comprising a test database. Let us return to the three-backend example, and integrate the information from Tables 33 and 37 for a small test database, with $N/4 = 74.976$ Mbytes.

Configuration 1 of Table 33 shows that we have 9,372 records for the 2000-byte record class. We wish to distribute these records according to the nine cluster categories of Table 37. Our task is to determine the entries for columns four, five, six, and seven of Table 38, such that the values for these entries produce a total of 9,372 2000-byte records.

TABLE 38. TARGET RECORD DISTRIBUTION TABLE.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2,000 | 2 | 4 | | | | |
| | 3 | 6 | | | | |
| | 4 | 8 | | | | |
| | 5 | 10 | | | | |
| | 6 | 12 | | | | |
| | 7 | 14 | | | | |
| | 8 | 16 | | | | |
| | 9 | 18 | | | | |
| | 10 | 20 | | | | |
| Sub-totals: | | 108 | | 9,372 | | |

To determine values for column four of Table 38, we first consider the mechanism used by MBDS to evenly distribute records across the backends. The insert information generation (IIG) process in the MBDS controller selects the backend at which the new record will be inserted. MBDS stores the records into blocks according to cluster-IDs. A new record is either added to a partially-filled block, or is inserted as the first record of a new block [Ref. 16: p. 51]. Thus, MBDS distributes blocks of records across the backends to achieve an even record distribution.

Let us consider a simple example to illustrate. We use the nine cluster categories and the corresponding values of the number of records per cluster category for the 2000-byte record class. We also assume a three-backend MBDS, with four clusters for each of the cluster categories. This results in the distribution shown in Table 39.

Given the distribution of Table 39, we want to see how MBDS distributes the blocks across three backends to effect an even record distribution. The first cluster category consists of two blocks per cluster, for four clusters, resulting in a total of eight blocks to be distributed across three backends. Since eight is not evenly divisible by three, MBDS distributes the blocks of this cluster in a {3,3,2} pattern. That is, two backends will receive three blocks of records, while one backend will receive two blocks of records. MBDS distributes the blocks for the rest of the clusters in a similar fashion. Table 40 shows the block and record distribution for this example.

TABLE 39. SAMPLE RECORD DISTRIBUTION.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks |
|---|---|---|---|---|---|
| 2,000 | 2 | 4 | 4 | 16 | 8 |
| | 3 | 6 | 4 | 24 | 12 |
| | 4 | 8 | 4 | 32 | 16 |
| | 5 | 10 | 4 | 40 | 20 |
| | 6 | 12 | 4 | 48 | 24 |
| | 7 | 14 | 4 | 56 | 28 |
| | 8 | 16 | 4 | 64 | 32 |
| | 9 | 18 | 4 | 72 | 36 |
| | 10 | 20 | 4 | 80 | 40 |
| Sub-totals: | | | 36 | 432 | 216 |

TABLE 40. RECORD/BLOCK DISTRIBUTION FOR TABLE 39 EXAMPLE.

| Cluster Category | Number of Blocks per Cluster | Backend #1 | | Backend #2 | | Backend #3 | |
|---|---|---|---|---|---|---|---|
| | | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 1 | 2 | 2 | 6 | 3 | 6 | 2 | 4 |
| 2 | 3 | 4 | 8 | 4 | 84 | 4 | 8 |
| 3 | 4 | 5 | 10 | 5 | 10 | 6 | 12 |
| 4 | 5 | 7 | 14 | 7 | 14 | 6 | 12 |
| 5 | 6 | 8 | 16 | 8 | 16 | 8 | 16 |
| 6 | 7 | 9 | 18 | 9 | 18 | 10 | 20 |
| 7 | 8 | 11 | 22 | 11 | 22 | 10 | 20 |
| 8 | 9 | 12 | 24 | 12 | 24 | 12 | 24 |
| 9 | 10 | 13 | 26 | 13 | 26 | 14 | 28 |
| Sub-totals: | | 72 | 144 | 72 | 144 | 72 | 144 |
| Note: (72 x 3) = 216 blocks. | | | | Note: (144 x 3) = 432 records. | | | |

Notice in Table 40 that during block distribution MBDS ensures that each backend ends up with an equal number of blocks. We observe that Backend #3 has received one less block for the first cluster category. During distribution of the blocks for the third cluster category, MBDS has compensated by inserting six blocks at Backend #3, while inserting only five blocks at Backends #1 and #2. The same situation occurs between cluster categories four and six, and between categories seven and nine of Table 40. Although it is not possible for the MBDS to distribute blocks equally for **every** individual cluster, it does work to achieve an equal distribution in the long run for the **entire** cluster collection.

With this understanding of the MBDS cluster distribution process, let us return to the task of determining the required number of clusters for column four of Table 38. Recall that the values we select must result in a total of 9,372 2000-byte records. If we sum column two of Table 38, we see that we have 108 records per cluster, distributed over all nine cluster categories. To develop a cluster distribution for column four of Table 38, we simply divide 9,372 by 108. The result is 86, with a remainder of 84. This means that we are 24, (108 - 84), records short of being able to use 87 clusters for each of the 9 cluster categories. This deficit is easily resolved by using 86 clusters for the first and last cluster categories, (since 4 + 20 = 24). The other seven categories will each have 87 clusters. The corresponding distribution is depicted in Table 41.

We may use the same values for the number of clusters shown in Table 41 to depict the record and block distribution of the 200, 400, and 1000-byte record

classes for Configuration 1 of Table 33, since 200, 400, and 1000 are all divisors of 2000. The resulting cluster distribution is shown in Table 42.

TABLE 41. RECORD/BLOCK DISTRIBUTION, SMALL DATABASE, CONFIGURATION 1.
(Based on the 2000-byte record class of Table 33)

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2,000 | 2 | 4 | 86 | 344 | 172 | 172 |
| | 3 | 6 | 87 | 522 | 261 | 261 |
| | 4 | 8 | 87 | 696 | 348 | 348 |
| | 5 | 10 | 87 | 870 | 435 | 435 |
| | 6 | 12 | 87 | 1,044 | 522 | 522 |
| | 7 | 14 | 87 | 1,218 | 609 | 609 |
| | 8 | 16 | 87 | 1,392 | 696 | 696 |
| | 9 | 18 | 87 | 1,566 | 783 | 783 |
| | 10 | 20 | 86 | 1,720 | 860 | 860 |
| Sub-totals: | | | 781 | 9,372 | 4,686 | 4,686 |

For Configuration 2 of Table 33, we must evenly distribute the database over two backends. We structure the database the same as depicted in Tables 41 and 42 for Configuration 1. The only change to be noted is that since the MBDS distributes the blocks evenly over the two backends, we must divide the values in column seven of Tables 41 and 42 by two to represent the number of blocks per backend for Configuration 2. The corresponding record/block distribution is shown in Table 43.

We have already seen an example of how MBDS distributes records over three backends when we referred to Tables 39 and 40. The corresponding distribution for Configuration 3 of Table 33 is depicted in Table 44.

For configuration 4, we see from Table 32 that the database for each backend has N Mbytes, so the total database size is 2N Mbytes. Thus, we must double the database size. The easiest way to do this is to double the number of blocks per cluster. In this way, we can easily double the number of records per cluster, while maintaining the same total number of clusters. Consider Tables 41 and 42, which show the database format for a database of size N, with one backend. To expand to a database of size 2N with a two backend system, we need to double the values in the first, second, fourth, and fifth columns of Tables 41 and 42. The result of this expansion is shown in Table 45.

For Configuration 5, we increase the database size to 3N Mbytes, and expand to three backends. Therefore, we need to triple the number of records per cluster, while maintaining the same total number of clusters. To achieve this, we triple all values in the first, second, fourth, and fifth columns of Tables 41 and 42. The result of this expansion is shown in Table 46.

TABLE 42. RECORD/BLOCK DISTRIBUTION, SMALL DATABASE, CONFIGURATION 1 (CONT'D).

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 1,000 | 2 | 8 | 86 | 688 | 172 | 172 |
| | 3 | 12 | 87 | 1,044 | 261 | 261 |
| | 4 | 16 | 87 | 1,392 | 348 | 348 |
| | 5 | 20 | 87 | 1,740 | 435 | 435 |
| | 6 | 24 | 87 | 2,088 | 522 | 522 |
| | 7 | 28 | 87 | 2,436 | 609 | 609 |
| | 8 | 32 | 87 | 2,784 | 696 | 696 |
| | 9 | 36 | 87 | 3,132 | 783 | 783 |
| | 10 | 40 | 86 | 3,440 | 860 | 860 |
| Sub-totals: | | | 781 | 18,744 | 4,686 | 4,686 |
| 400 | 2 | 20 | 86 | 1,720 | 172 | 172 |
| | 3 | 30 | 87 | 2,610 | 261 | 261 |
| | 4 | 40 | 87 | 3,480 | 348 | 348 |
| | 5 | 50 | 87 | 4,350 | 435 | 435 |
| | 6 | 60 | 87 | 5,220 | 522 | 522 |
| | 7 | 70 | 87 | 6,090 | 609 | 609 |
| | 8 | 80 | 87 | 6,960 | 696 | 696 |
| | 9 | 90 | 87 | 7,830 | 783 | 783 |
| | 10 | 100 | 86 | 8,600 | 860 | 860 |
| Sub-totals: | | | 781 | 46,860 | 4,686 | 4,686 |
| 200 | 2 | 40 | 86 | 3,440 | 172 | 172 |
| | 3 | 60 | 87 | 5,220 | 261 | 261 |
| | 4 | 80 | 87 | 6,960 | 348 | 348 |
| | 5 | 100 | 87 | 8,700 | 435 | 435 |
| | 6 | 120 | 87 | 10,440 | 522 | 522 |
| | 7 | 140 | 87 | 12,180 | 609 | 609 |
| | 8 | 160 | 87 | 13,920 | 696 | 696 |
| | 9 | 180 | 87 | 15,660 | 783 | 783 |
| | 10 | 200 | 86 | 17,200 | 860 | 860 |
| Sub-totals: | | | 781 | 93,720 | 4,686 | 4,686 |

TABLE 43. RECORD BLOCK DISTRIBUTION.
SMALL DATABASE, CONFIGURATION 2.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Backend 1 | Backend 2 |
|---|---|---|---|---|---|---|---|
| | | | | | | Number of Blocks | Number of Blocks |
| 2000 | 2 | 4 | 86 | 344 | 172 | 86 | 86 |
| | 3 | 6 | 87 | 522 | 261 | 131 | 130 |
| | 4 | 8 | 87 | 696 | 348 | 174 | 174 |
| | 5 | 10 | 87 | 870 | 435 | 217 | 218 |
| | 6 | 12 | 87 | 1,044 | 522 | 261 | 261 |
| | 7 | 14 | 87 | 1,218 | 609 | 305 | 304 |
| | 8 | 16 | 87 | 1,392 | 696 | 348 | 348 |
| | 9 | 18 | 87 | 1,566 | 783 | 391 | 392 |
| | 10 | 20 | 86 | 1,720 | 860 | 430 | 430 |
| Sub-totals: | | | 781 | 9,372 | 4,686 | 2,343 | 2,343 |
| 1000 | 2 | 8 | 86 | 688 | 172 | 86 | 86 |
| | 3 | 12 | 87 | 1,044 | 261 | 131 | 130 |
| | 4 | 16 | 87 | 1,392 | 348 | 174 | 174 |
| | 5 | 20 | 87 | 1,740 | 435 | 217 | 218 |
| | 6 | 24 | 87 | 2,088 | 522 | 261 | 261 |
| | 7 | 28 | 87 | 2,436 | 609 | 305 | 304 |
| | 8 | 32 | 87 | 2,784 | 696 | 348 | 348 |
| | 9 | 36 | 87 | 3,132 | 783 | 391 | 392 |
| | 10 | 40 | 86 | 3,440 | 860 | 430 | 430 |
| Sub-totals: | | | 781 | 18,744 | 4,686 | 2,343 | 2,343 |
| 400 | 2 | 20 | 86 | 1,720 | 172 | 86 | 86 |
| | 3 | 30 | 87 | 2,610 | 261 | 131 | 130 |
| | 4 | 40 | 87 | 3,480 | 348 | 174 | 174 |
| | 5 | 50 | 87 | 4,350 | 435 | 217 | 218 |
| | 6 | 60 | 87 | 5,220 | 522 | 261 | 261 |
| | 7 | 70 | 87 | 6,090 | 609 | 305 | 304 |
| | 8 | 80 | 87 | 6,960 | 696 | 348 | 348 |
| | 9 | 90 | 87 | 7,830 | 783 | 391 | 392 |
| | 10 | 100 | 86 | 8,600 | 860 | 430 | 430 |
| Sub-totals: | | | 781 | 46,860 | 4,686 | 2,343 | 2,343 |
| 200 | 2 | 40 | 86 | 3,440 | 172 | 86 | 86 |
| | 3 | 60 | 87 | 5,220 | 261 | 131 | 130 |
| | 4 | 80 | 87 | 6,960 | 348 | 174 | 174 |
| | 5 | 100 | 87 | 8,700 | 435 | 217 | 218 |
| | 6 | 120 | 87 | 10,440 | 522 | 261 | 261 |
| | 7 | 140 | 87 | 12,180 | 609 | 305 | 304 |
| | 8 | 160 | 87 | 13,920 | 696 | 348 | 348 |
| | 9 | 180 | 87 | 15,660 | 783 | 391 | 392 |
| | 10 | 200 | 86 | 17,200 | 860 | 430 | 430 |
| Sub-totals: | | | 781 | 93,720 | 4,686 | 2,343 | 2,343 |

TABLE 44a. RECORD/BLOCK DISTRIBUTION. SMALL DATABASE. CONFIGURATION 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2,000 | 2 | 4 | 86 | 344 | 172 | |
| | 3 | 6 | 87 | 522 | 261 | |
| | 4 | 8 | 87 | 696 | 348 | (See |
| | 5 | 10 | 87 | 870 | 435 | below) |
| | 6 | 12 | 87 | 1,044 | 522 | |
| | 7 | 14 | 87 | 1,218 | 609 | |
| | 8 | 16 | 87 | 1,392 | 696 | |
| | 9 | 18 | 87 | 1,566 | 783 | |
| | 10 | 20 | 86 | 1,720 | 860 | |
| Sub-totals: | | | 781 | 9,372 | 4,686 | |

| Number of Blocks per Cluster | Backend #1 | | Backend #2 | | Backend #3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 58 | 116 | 57 | 114 | 57 | 114 |
| 3 | 87 | 174 | 87 | 174 | 87 | 174 |
| 4 | 116 | 232 | 116 | 232 | 116 | 232 |
| 5 | 145 | 290 | 145 | 290 | 145 | 290 |
| 6 | 174 | 348 | 174 | 348 | 174 | 348 |
| 7 | 203 | 406 | 203 | 406 | 203 | 406 |
| 8 | 232 | 464 | 232 | 464 | 232 | 464 |
| 9 | 261 | 522 | 261 | 522 | 261 | 522 |
| 10 | 286 | 572 | 287 | 574 | 287 | 574 |
| Sub-totals | 1,562 | 3,124 | 1,562 | 3,124 | 1,562 | 3,124 |

TABLE 44b. RECORD/BLOCK DISTRIBUTION, SMALL DATABASE, CONFIGURATION 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 1,000 | 2 | 8 | 86 | 688 | 172 | |
| | 3 | 12 | 87 | 1,044 | 261 | |
| | 4 | 16 | 87 | 1,392 | 348 | (See |
| | 5 | 20 | 87 | 1,740 | 435 | below) |
| | 6 | 24 | 87 | 2,088 | 522 | |
| | 7 | 28 | 87 | 2,436 | 609 | |
| | 8 | 32 | 87 | 2,784 | 696 | |
| | 9 | 36 | 87 | 3,132 | 783 | |
| | 10 | 40 | 86 | 3,440 | 860 | |
| Sub-totals: | | | 781 | 18,744 | 4,686 | |

| Number of Blocks per Cluster | Backend #1 | | Backend #2 | | Backend #3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 58 | 232 | 57 | 228 | 57 | 228 |
| 3 | 87 | 348 | 87 | 348 | 87 | 348 |
| 4 | 116 | 464 | 116 | 464 | 116 | 464 |
| 5 | 145 | 580 | 145 | 580 | 145 | 580 |
| 6 | 174 | 696 | 174 | 696 | 174 | 696 |
| 7 | 203 | 812 | 203 | 812 | 203 | 812 |
| 8 | 232 | 928 | 232 | 928 | 232 | 928 |
| 9 | 261 | 1,044 | 261 | 1,044 | 261 | 1,044 |
| 10 | 286 | 1,144 | 287 | 1,148 | 287 | 1,148 |
| Sub-totals: | 1,562 | 6,248 | 1,562 | 6,248 | 1,562 | 6,248 |

TABLE 44c. RECORD/BLOCK DISTRIBUTION, SMALL DATABASE, CONFIGURATION 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 400 | 2 | 20 | 86 | 1,720 | 172 | |
| | 3 | 30 | 87 | 2,610 | 261 | |
| | 4 | 40 | 87 | 3,480 | 348 | (See |
| | 5 | 50 | 87 | 4,350 | 435 | below) |
| | 6 | 60 | 87 | 5,220 | 522 | |
| | 7 | 70 | 87 | 6,090 | 609 | |
| | 8 | 80 | 87 | 6,960 | 696 | |
| | 9 | 90 | 87 | 7,830 | 783 | |
| | 10 | 100 | 86 | 8,600 | 860 | |
| Sub-totals: | | | 781 | 46,860 | 4,686 | |

| Number of Blocks per Cluster | Backend #1 | | Backend #2 | | Backend #3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 58 | 580 | 57 | 570 | 57 | 570 |
| 3 | 87 | 870 | 87 | 870 | 87 | 870 |
| 4 | 116 | 1,160 | 116 | 1,160 | 116 | 1,160 |
| 5 | 145 | 1,450 | 145 | 1,450 | 145 | 1,450 |
| 6 | 174 | 1,740 | 174 | 1,740 | 174 | 1,740 |
| 7 | 203 | 2,030 | 203 | 2,030 | 203 | 2,030 |
| 8 | 232 | 2,320 | 232 | 2,320 | 232 | 2,320 |
| 9 | 261 | 2,610 | 261 | 2,610 | 261 | 2,610 |
| 10 | 286 | 2,860 | 287 | 2,860 | 287 | 2,860 |
| Sub-totals: | 1,562 | 15,620 | 1,562 | 15,620 | 1,562 | 15,620 |

TABLE 44d. RECORD/BLOCK DISTRIBUTION, SMALL DATABASE, CONFIGURATION 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 200 | 2 | 40 | 86 | 3,440 | 172 | |
| | 3 | 60 | 87 | 5,220 | 261 | |
| | 4 | 80 | 87 | 6,960 | 348 | (See |
| | 5 | 100 | 87 | 8,700 | 435 | below) |
| | 6 | 120 | 87 | 10,440 | 522 | |
| | 7 | 140 | 87 | 12,180 | 609 | |
| | 8 | 160 | 87 | 13,920 | 696 | |
| | 9 | 180 | 87 | 15,660 | 783 | |
| | 10 | 200 | 86 | 17,200 | 860 | |
| Sub-totals: | | | 781 | 93,720 | 4,686 | |

| Number of Blocks per Cluster | Backend #1 | | Backend #2 | | Backend #3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 58 | 1,160 | 57 | 1,140 | 57 | 1,140 |
| 3 | 87 | 1,740 | 87 | 1,740 | 87 | 1,740 |
| 4 | 116 | 2,320 | 116 | 2,320 | 116 | 2,320 |
| 5 | 145 | 2,900 | 145 | 2,900 | 145 | 2,900 |
| 6 | 174 | 3,480 | 174 | 3,480 | 174 | 3,480 |
| 7 | 203 | 4,060 | 203 | 4,060 | 203 | 4,060 |
| 8 | 232 | 4,640 | 232 | 4,640 | 232 | 4,640 |
| 9 | 261 | 5,220 | 261 | 5,220 | 261 | 5,220 |
| 10 | 286 | 5,720 | 287 | 5,740 | 287 | 5,740 |
| Sub-totals: | 1,562 | 31,240 | 1,562 | 31,240 | 1,562 | 31,240 |

TABLE 45. RECORD BLOCK DISTRIBUTION.
SMALL DATABASE. CONFIGURATION 4.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 4 | 8 | 86 | 688 | 344 | 172 |
| | 6 | 12 | 87 | 1,044 | 522 | 261 |
| | 8 | 16 | 87 | 1,392 | 696 | 348 |
| | 10 | 20 | 87 | 1,740 | 870 | 435 |
| | 12 | 24 | 87 | 2,088 | 1,044 | 522 |
| | 14 | 28 | 87 | 2,436 | 1,218 | 609 |
| | 16 | 32 | 87 | 2,784 | 1,392 | 696 |
| | 18 | 36 | 87 | 3,132 | 1,566 | 783 |
| | 20 | 40 | 86 | 3,440 | 1,720 | 860 |
| Sub-totals: | | | 781 | 18,744 | 9,372 | 4,686 |
| 1000 | 4 | 16 | 86 | 1,376 | 344 | 172 |
| | 6 | 24 | 87 | 2,088 | 522 | 261 |
| | 8 | 32 | 87 | 2,784 | 696 | 348 |
| | 10 | 40 | 87 | 3,480 | 870 | 435 |
| | 12 | 48 | 87 | 4,176 | 1,044 | 522 |
| | 14 | 56 | 87 | 4,872 | 1,218 | 609 |
| | 16 | 64 | 87 | 5,568 | 1,392 | 696 |
| | 18 | 72 | 87 | 6,264 | 1,566 | 783 |
| | 20 | 80 | 86 | 6,880 | 1,720 | 860 |
| Sub-totals: | | | 781 | 37,488 | 9,372 | 4,686 |
| 400 | 4 | 40 | 86 | 3,440 | 344 | 172 |
| | 6 | 60 | 87 | 5,220 | 522 | 261 |
| | 8 | 80 | 87 | 6,960 | 696 | 348 |
| | 10 | 100 | 87 | 8,700 | 870 | 435 |
| | 12 | 120 | 87 | 10,440 | 1,044 | 522 |
| | 14 | 140 | 87 | 12,180 | 1,218 | 609 |
| | 16 | 160 | 87 | 13,920 | 1,392 | 696 |
| | 18 | 180 | 87 | 15,660 | 1,566 | 783 |
| | 20 | 200 | 86 | 17,200 | 1,720 | 860 |
| Sub-totals: | | | 781 | 93,720 | 9,372 | 4,686 |
| 2000 | 4 | 80 | 86 | 6,880 | 344 | 172 |
| | 6 | 120 | 87 | 10,440 | 522 | 261 |
| | 8 | 160 | 87 | 13,920 | 696 | 348 |
| | 10 | 200 | 87 | 17,400 | 870 | 435 |
| | 12 | 240 | 87 | 20,880 | 1,044 | 522 |
| | 14 | 280 | 87 | 24,360 | 1,218 | 609 |
| | 16 | 320 | 87 | 27,840 | 1,392 | 696 |
| | 18 | 360 | 87 | 31,320 | 1,566 | 783 |
| | 20 | 400 | 86 | 34,400 | 1,720 | 860 |
| Sub-totals: | | | 781 | 187,440 | 9,372 | 4,686 |

TABLE 46. RECORD BLOCK DISTRIBUTION.
SMALL DATABASE. CONFIGURATION 5.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 6 | 12 | 86 | 1,032 | 516 | 172 |
| | 9 | 18 | 87 | 1,566 | 783 | 261 |
| | 12 | 24 | 87 | 2,088 | 1,044 | 348 |
| | 15 | 30 | 87 | 2,610 | 1,305 | 435 |
| | 18 | 36 | 87 | 3,132 | 1,566 | 522 |
| | 21 | 42 | 87 | 3,654 | 1,827 | 609 |
| | 24 | 48 | 87 | 4,176 | 2,088 | 696 |
| | 27 | 54 | 87 | 4,698 | 2,349 | 783 |
| | 30 | 60 | 86 | 5,160 | 2,580 | 860 |
| Sub-totals: | | | 781 | 28,116 | 14,058 | 4,686 |
| 1000 | 6 | 24 | 86 | 2,064 | 516 | 172 |
| | 9 | 36 | 87 | 3,132 | 783 | 261 |
| | 12 | 48 | 87 | 4,176 | 1,044 | 348 |
| | 15 | 60 | 87 | 5,220 | 1,305 | 435 |
| | 18 | 72 | 87 | 6,264 | 1,566 | 522 |
| | 21 | 84 | 87 | 7,308 | 1,827 | 609 |
| | 24 | 96 | 87 | 8,352 | 2,088 | 696 |
| | 27 | 108 | 87 | 9.396 | 2,349 | 783 |
| | 30 | 120 | 86 | 10,320 | 2,580 | 860 |
| Sub-totals: | | | 781 | 56,232 | 14,058 | 4,686 |
| 400 | 6 | 60 | 86 | 5,160 | 516 | 172 |
| | 9 | 90 | 87 | 7,830 | 783 | 261 |
| | 12 | 120 | 87 | 10,440 | 1,044 | 348 |
| | 15 | 150 | 87 | 13,050 | 1,305 | 435 |
| | 18 | 180 | 87 | 15,660 | 1,566 | 522 |
| | 21 | 210 | 87 | 18,270 | 1,827 | 609 |
| | 24 | 240 | 87 | 20,880 | 2,088 | 696 |
| | 27 | 270 | 87 | 23,490 | 2,349 | 783 |
| | 30 | 300 | 86 | 25,800 | 2,580 | 860 |
| Sub-totals: | | | 781 | 140,580 | 14,058 | 4,686 |
| 200 | 6 | 120 | 86 | 10,320 | 516 | 172 |
| | 9 | 180 | 87 | 15,660 | 783 | 261 |
| | 12 | 240 | 87 | 20,880 | 1,044 | 348 |
| | 15 | 300 | 87 | 26,100 | 1,305 | 435 |
| | 18 | 360 | 87 | 31,320 | 1,566 | 522 |
| | 21 | 420 | 87 | 36,540 | 1,827 | 609 |
| | 24 | 480 | 87 | 41,760 | 2,088 | 696 |
| | 27 | 540 | 87 | 46,980 | 2,349 | 783 |
| | 30 | 600 | 86 | 51,600 | 2,580 | 860 |
| Sub-totals: | | | 781 | 281,160 | 14,058 | 4,686 |

Next, we consider the configurations for the medium-size database, with N/2 = 149.952 Mbytes. Since this database is twice as big as the small database we have just described, we now have twice as many records as depicted in Tables 41 and 42. To create twice as many records, we double the number of clusters in column three of Table 41, while keeping the number of blocks per cluster, and the number of records per cluster the same as shown in columns one and two of Tables 41/42. The corresponding database format for Configuration 1 with a medium-size database is shown in Table 47.

The record/block distributions for configurations 2, 3, 4, and 5 of Table 34 for the medium database case are developed in an identical manner as described previously for the corresponding configurations of Table 33, except that we now use Table 47 as our base table, instead of Tables 41 and 42. The record/block distributions for configurations 2, 3, 4, and 5 for the medium-size database are shown in Tables 48 through 51.

Finally, we consider the configurations for the large database depicted in Table 35, with N = 299.904 Mbytes. Since this database is four times as large as the small database we have described in Tables 41 through 46, we now have four times as many records as we have depicted in Tables 41 and 42. To create four times as many records, we quadruple the number of clusters in column three of Table 41. The corresponding record/block distributions for configurations 1-5 of Table 35 are shown in Tables 52 through 56.

## B. RECORD TEMPLATES AND DESCRIPTOR DEFINITIONS

To complete the development of the MBDS test-database set, we must specify the directory structure for the specific record types that are used. This requires that we define record templates for each of the four record classes to specify the record structures we will be using. A **record template** is the formal specification of the directory and non-directory attributes which make up the actual record structure and determine the intended directory descriptors [Ref. 10: p. 10]. In this section we define the required record templates, and then describe the descriptor types and descriptor ranges for the corresponding directory attributes.

TABLE 47. RECORD/BLOCK DISTRIBUTION, MEDIUM DATABASE, CONFIGURATION 1.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 2 | 4 | 172 | 688 | 344 | 344 |
| | 3 | 6 | 174 | 1,044 | 522 | 522 |
| | 4 | 8 | 174 | 1,392 | 696 | 696 |
| | 5 | 10 | 174 | 1,740 | 870 | 870 |
| | 6 | 12 | 174 | 2,088 | 1,044 | 1,044 |
| | 7 | 14 | 174 | 2,436 | 1,218 | 1,218 |
| | 8 | 16 | 174 | 2,784 | 1,392 | 1,392 |
| | 9 | 18 | 174 | 3,132 | 1,566 | 1,566 |
| | 10 | 20 | 172 | 3,440 | 1,720 | 1,720 |
| Sub-totals: | | | 1,562 | 18,744 | 9,372 | 9,372 |
| 1000 | 2 | 8 | 172 | 1,376 | 344 | 344 |
| | 3 | 12 | 174 | 2,088 | 522 | 522 |
| | 4 | 16 | 174 | 2,784 | 696 | 696 |
| | 5 | 20 | 174 | 3,480 | 870 | 870 |
| | 6 | 24 | 174 | 4,176 | 1,044 | 1,044 |
| | 7 | 28 | 174 | 4,872 | 1,218 | 1,218 |
| | 8 | 32 | 174 | 5,568 | 1,392 | 1,392 |
| | 9 | 36 | 174 | 6,264 | 1,566 | 1,566 |
| | 10 | 40 | 172 | 6,880 | 1,720 | 1,720 |
| Sub-totals: | | | 1,562 | 37,488 | 9,372 | 9,372 |
| 400 | 2 | 20 | 172 | 3,440 | 344 | 344 |
| | 3 | 30 | 174 | 5,220 | 522 | 522 |
| | 4 | 40 | 174 | 6,960 | 696 | 696 |
| | 5 | 50 | 174 | 8,700 | 870 | 870 |
| | 6 | 60 | 174 | 10,440 | 1,044 | 1,044 |
| | 7 | 70 | 174 | 12,180 | 1,218 | 1,218 |
| | 8 | 80 | 174 | 13,920 | 1,392 | 1,392 |
| | 9 | 90 | 174 | 15,660 | 1,566 | 1,566 |
| | 10 | 100 | 172 | 17,200 | 1,720 | 1,720 |
| Sub-totals: | | | 1,562 | 93,720 | 9,372 | 9,372 |
| 200 | 2 | 40 | 172 | 6,880 | 344 | 344 |
| | 3 | 60 | 174 | 10,440 | 522 | 522 |
| | 4 | 80 | 174 | 13,920 | 696 | 696 |
| | 5 | 100 | 174 | 17,400 | 870 | 870 |
| | 6 | 120 | 174 | 20,880 | 1,044 | 1,044 |
| | 7 | 140 | 174 | 24,360 | 1,218 | 1,218 |
| | 8 | 160 | 174 | 27,840 | 1,392 | 1,392 |
| | 9 | 180 | 174 | 31,320 | 1,566 | 1,566 |
| | 10 | 200 | 172 | 34,400 | 1,720 | 1,720 |
| Sub-totals: | | | 1,562 | 187,440 | 9,372 | 9,372 |

TABLE 48. RECORD/BLOCK DISTRIBUTION, MEDIUM DATABASE, CONFIGURATION 2.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 2 | 4 | 172 | 688 | 344 | 172 |
| | 3 | 6 | 174 | 1,044 | 522 | 261 |
| | 4 | 8 | 174 | 1,392 | 696 | 348 |
| | 5 | 10 | 174 | 1,740 | 870 | 435 |
| | 6 | 12 | 174 | 2,088 | 1,044 | 522 |
| | 7 | 14 | 174 | 2,436 | 1,218 | 609 |
| | 8 | 16 | 174 | 2,784 | 1,392 | 696 |
| | 9 | 18 | 174 | 3,132 | 1,566 | 783 |
| | 10 | 20 | 172 | 3,440 | 1,720 | 860 |
| Sub-totals: | | | 1,562 | 18,744 | 9,372 | 4,686 |
| 1000 | 2 | 8 | 172 | 1,376 | 344 | 172 |
| | 3 | 12 | 174 | 2,088 | 522 | 261 |
| | 4 | 16 | 174 | 2,784 | 696 | 348 |
| | 5 | 20 | 174 | 3,480 | 870 | 435 |
| | 6 | 24 | 174 | 4,176 | 1,044 | 522 |
| | 7 | 28 | 174 | 4,872 | 1,218 | 609 |
| | 8 | 32 | 174 | 5,568 | 1,392 | 696 |
| | 9 | 36 | 174 | 6,264 | 1,566 | 783 |
| | 10 | 40 | 172 | 6,880 | 1,720 | 860 |
| Sub-totals: | | | 1,562 | 37,488 | 9,372 | 4,686 |
| 400 | 2 | 20 | 172 | 3,440 | 344 | 172 |
| | 3 | 30 | 174 | 5,220 | 522 | 261 |
| | 4 | 40 | 174 | 6,960 | 696 | 348 |
| | 5 | 50 | 174 | 8,700 | 870 | 435 |
| | 6 | 60 | 174 | 10,440 | 1,044 | 522 |
| | 7 | 70 | 174 | 12,180 | 1,218 | 609 |
| | 8 | 80 | 174 | 13,920 | 1,392 | 696 |
| | 9 | 90 | 174 | 15,660 | 1,566 | 783 |
| | 10 | 100 | 172 | 17,200 | 1,720 | 860 |
| Sub-totals: | | | 1,562 | 93,720 | 9,372 | 4,686 |
| 200 | 2 | 40 | 172 | 6,880 | 344 | 172 |
| | 3 | 60 | 174 | 10,440 | 522 | 261 |
| | 4 | 80 | 174 | 13,920 | 696 | 348 |
| | 5 | 100 | 174 | 17,400 | 870 | 435 |
| | 6 | 120 | 174 | 20,880 | 1,044 | 522 |
| | 7 | 140 | 174 | 24,360 | 1,218 | 609 |
| | 8 | 160 | 174 | 27,840 | 1,392 | 696 |
| | 9 | 180 | 174 | 31,320 | 1,566 | 783 |
| | 10 | 200 | 172 | 34,400 | 1,720 | 860 |
| Sub-totals: | | | 1,562 | 187,440 | 9,372 | 4,686 |

TABLE 49a. RECORD/BLOCK DISTRIBUTION, MEDIUM DATABASE. CONFIGURATION 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 2 | 4 | 172 | 688 | 344 | |
| | 3 | 6 | 174 | 1,044 | 522 | |
| | 4 | 8 | 174 | 1,392 | 696 | (See |
| | 5 | 10 | 174 | 1,740 | 870 | below) |
| | 6 | 12 | 174 | 2,088 | 1,044 | |
| | 7 | 14 | 174 | 2,436 | 1,218 | |
| | 8 | 16 | 174 | 2,784 | 1,392 | |
| | 9 | 18 | 174 | 3,132 | 1,566 | |
| | 10 | 20 | 172 | 3,440 | 1,720 | |
| Sub-totals: | | | 1,562 | 18,744 | 9,372 | |

| Number of Blocks per Cluster | Backend #1 | | Backend #2 | | Backend #3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 116 | 232 | 114 | 228 | 114 | 228 |
| 3 | 174 | 348 | 174 | 348 | 174 | 348 |
| 4 | 232 | 464 | 232 | 464 | 232 | 464 |
| 5 | 290 | 580 | 290 | 580 | 290 | 580 |
| 6 | 348 | 696 | 348 | 696 | 348 | 696 |
| 7 | 406 | 812 | 406 | 812 | 406 | 812 |
| 8 | 464 | 928 | 464 | 928 | 464 | 928 |
| 9 | 522 | 1,044 | 522 | 1,044 | 522 | 1,044 |
| 10 | 572 | 1,144 | 574 | 1,148 | 574 | 1,148 |
| Sub-totals: | 3,124 | 6,248 | 3,124 | 6,248 | 3,124 | 6,248 |

TABLE 49b. RECORD/BLOCK DISTRIBUTION, MEDIUM DATABASE. CONFIGURATION 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 1000 | 2 | 8 | 172 | 1,376 | 344 | |
| | 3 | 12 | 174 | 2,088 | 522 | |
| | 4 | 16 | 174 | 2,784 | 696 | (See |
| | 5 | 20 | 174 | 3,480 | 870 | below) |
| | 6 | 24 | 174 | 4,176 | 1,044 | |
| | 7 | 28 | 174 | 4,872 | 1,218 | |
| | 8 | 32 | 174 | 5,568 | 1,392 | |
| | 9 | 36 | 174 | 6,264 | 1,566 | |
| | 10 | 40 | 172 | 6,880 | 1,720 | |
| Sub-totals: | | | 1,562 | 37,488 | 9,372 | |

| Number of Blocks per Cluster | Backend #1 | | Backend #2 | | Backend #3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 116 | 464 | 114 | 456 | 114 | 456 |
| 3 | 174 | 696 | 174 | 696 | 174 | 696 |
| 4 | 232 | 928 | 232 | 928 | 232 | 928 |
| 5 | 290 | 1,160 | 290 | 1,160 | 290 | 1,160 |
| 6 | 348 | 1,392 | 348 | 1,392 | 348 | 1,392 |
| 7 | 406 | 1,624 | 406 | 1,624 | 406 | 1,624 |
| 8 | 464 | 1,856 | 464 | 1,856 | 464 | 1,856 |
| 9 | 522 | 2,088 | 522 | 2,088 | 522 | 2,088 |
| 10 | 572 | 2,288 | 574 | 2,296 | 574 | 2,296 |
| Sub-totals: | 3,124 | 12,496 | 3,124 | 12,496 | 3,124 | 12,496 |

TABLE 49c. RECORD/BLOCK DISTRIBUTION. MEDIUM DATABASE, CONFIGURATION 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 400 | 2 | 20 | 172 | 3,440 | 344 | |
| | 3 | 30 | 174 | 5,220 | 522 | |
| | 4 | 40 | 174 | 6,960 | 696 | (See |
| | 5 | 50 | 174 | 8,700 | 870 | below) |
| | 6 | 60 | 174 | 10,440 | 1,044 | |
| | 7 | 70 | 174 | 12,180 | 1,218 | |
| | 8 | 80 | 174 | 13,920 | 1,392 | |
| | 9 | 90 | 174 | 15,660 | 1,566 | |
| | 10 | 100 | 172 | 17,200 | 1,720 | |
| Sub-totals: | | | 1,562 | 93,720 | 9,372 | |

| Number of Blocks per Cluster | Backend #1 | | Backend #2 | | Backend #3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 116 | 1,160 | 114 | 1,140 | 114 | 1,140 |
| 3 | 174 | 1,740 | 174 | 1,740 | 174 | 1,740 |
| 4 | 232 | 2,320 | 232 | 2,320 | 232 | 2,320 |
| 5 | 290 | 2,900 | 290 | 2,900 | 290 | 2,900 |
| 6 | 348 | 3,480 | 348 | 3,480 | 348 | 3,480 |
| 7 | 406 | 4,060 | 406 | 4,060 | 406 | 4,060 |
| 8 | 464 | 4,640 | 464 | 4,640 | 464 | 4,640 |
| 9 | 522 | 5,220 | 522 | 5,220 | 522 | 5,220 |
| 10 | 572 | 5,720 | 574 | 5,740 | 574 | 5,740 |
| Sub-totals: | 3,124 | 31,240 | 3,124 | 31,240 | 3,124 | 31,240 |

TABLE 49d. RECORD/BLOCK DISTRIBUTION, MEDIUM DATABASE, CONFIGURATION 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 200 | 2 | 40 | 172 | 6,880 | 344 | |
| | 3 | 60 | 174 | 10,440 | 522 | |
| | 4 | 80 | 174 | 13,920 | 696 | (See |
| | 5 | 100 | 174 | 17,400 | 870 | below) |
| | 6 | 120 | 174 | 20,880 | 1,044 | |
| | 7 | 140 | 174 | 24,360 | 1,218 | |
| | 8 | 160 | 174 | 27,840 | 1,392 | |
| | 9 | 180 | 174 | 31,320 | 1,566 | |
| | 10 | 200 | 172 | 34,400 | 1,720 | |
| Sub-totals: | | | 1,562 | 187,440 | 9,372 | |

| Number of Blocks per Cluster | Backend #1 | | Backend #2 | | Backend #3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 116 | 2,320 | 114 | 2,280 | 114 | 2,280 |
| 3 | 174 | 3,480 | 174 | 3,480 | 174 | 3,480 |
| 4 | 232 | 4,640 | 232 | 4,640 | 232 | 4,640 |
| 5 | 290 | 5,800 | 290 | 5,800 | 290 | 5,800 |
| 6 | 348 | 6,960 | 348 | 6,960 | 348 | 6,960 |
| 7 | 406 | 8,120 | 406 | 8,120 | 406 | 8,120 |
| 8 | 464 | 9,280 | 464 | 9,280 | 464 | 9,280 |
| 9 | 522 | 10,440 | 522 | 10,440 | 522 | 10,440 |
| 10 | 572 | 11,440 | 574 | 11,480 | 574 | 11,480 |
| Sub-totals: | 3,124 | 62,480 | 3,124 | 62,480 | 3,124 | 62,480 |

TABLE 50. RECORD/BLOCK DISTRIBUTION.
MEDIUM DATABASE. CONFIGURATION 4.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 4 | 8 | 172 | 1,376 | 688 | 344 |
| | 6 | 12 | 174 | 2,088 | 1,044 | 522 |
| | 8 | 16 | 174 | 2,784 | 1,392 | 696 |
| | 10 | 20 | 174 | 3,480 | 1,740 | 870 |
| | 12 | 24 | 174 | 4,176 | 2,088 | 1,044 |
| | 14 | 28 | 174 | 4,872 | 2,436 | 1,218 |
| | 16 | 32 | 174 | 5,568 | 2,784 | 1,392 |
| | 18 | 36 | 174 | 6,264 | 3,132 | 1,566 |
| | 20 | 40 | 172 | 6,880 | 3,440 | 1,720 |
| Sub-totals: | | | 1,562 | 37,488 | 18,744 | 9,372 |
| 1000 | 4 | 16 | 172 | 2,752 | 688 | 344 |
| | 6 | 24 | 174 | 4,176 | 1,044 | 522 |
| | 8 | 32 | 174 | 5,568 | 1,392 | 696 |
| | 10 | 40 | 174 | 6,960 | 1,740 | 870 |
| | 12 | 48 | 174 | 8,352 | 2,088 | 1,044 |
| | 14 | 56 | 174 | 9,744 | 2,436 | 1,218 |
| | 16 | 64 | 174 | 11,136 | 2,784 | 1,392 |
| | 18 | 72 | 174 | 12,528 | 3,132 | 1,566 |
| | 20 | 80 | 172 | 13,760 | 3,440 | 1,720 |
| Sub-totals: | | | 1,562 | 74,976 | 18,744 | 9,372 |
| 400 | 4 | 40 | 172 | 6,880 | 688 | 344 |
| | 6 | 60 | 174 | 10,440 | 1,044 | 522 |
| | 8 | 80 | 174 | 13,920 | 1,392 | 696 |
| | 10 | 100 | 174 | 17,400 | 1,740 | 870 |
| | 12 | 120 | 174 | 20,880 | 2,088 | 1,044 |
| | 14 | 140 | 174 | 24,360 | 2,436 | 1,218 |
| | 16 | 160 | 174 | 27,840 | 2,784 | 1,392 |
| | 18 | 180 | 174 | 31,320 | 3,132 | 1,566 |
| | 20 | 200 | 172 | 34,400 | 3,440 | 1,720 |
| Sub-totals: | | | 1,562 | 187,440 | 18,744 | 9,372 |
| 200 | 4 | 80 | 172 | 13,760 | 688 | 344 |
| | 6 | 120 | 174 | 20,880 | 1,044 | 522 |
| | 8 | 160 | 174 | 27,840 | 1,392 | 696 |
| | 10 | 200 | 174 | 34,800 | 1,740 | 870 |
| | 12 | 240 | 174 | 41,760 | 2,088 | 1,044 |
| | 14 | 280 | 174 | 48,720 | 2,436 | 1,218 |
| | 16 | 320 | 174 | 55,680 | 2,784 | 1,392 |
| | 18 | 360 | 174 | 62,640 | 3,132 | 1,566 |
| | 20 | 400 | 172 | 68,800 | 3,440 | 1,720 |
| Sub-totals: | | | 1,562 | 374,880 | 18,744 | 9,372 |

TABLE 51. RECORD/BLOCK DISTRIBUTION.
MEDIUM DATABASE, CONFIGURATION 5.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 6 | 12 | 172 | 2,064 | 1,032 | 344 |
| | 9 | 18 | 174 | 3,132 | 1,566 | 522 |
| | 12 | 24 | 174 | 4,176 | 2,088 | 696 |
| | 15 | 30 | 174 | 5,220 | 2,610 | 870 |
| | 18 | 36 | 174 | 6,264 | 3,132 | 1,044 |
| | 21 | 42 | 174 | 7,308 | 3,654 | 1,218 |
| | 24 | 48 | 174 | 8,352 | 4,176 | 1,392 |
| | 27 | 54 | 174 | 9,396 | 4,698 | 1,566 |
| | 30 | 60 | 172 | 10,320 | 5,160 | 1,720 |
| Sub-totals: | | | 1,562 | 56,232 | 28,116 | 9,372 |
| 1000 | 6 | 24 | 172 | 4,128 | 1,032 | 344 |
| | 9 | 36 | 174 | 6,264 | 1,566 | 522 |
| | 12 | 48 | 174 | 8,352 | 2,088 | 696 |
| | 15 | 60 | 174 | 10,440 | 2,610 | 870 |
| | 18 | 72 | 174 | 12,528 | 3,132 | 1,044 |
| | 21 | 84 | 174 | 14,616 | 3,654 | 1,218 |
| | 24 | 96 | 174 | 16,704 | 4,176 | 1,392 |
| | 27 | 108 | 174 | 18,792 | 4,698 | 1,566 |
| | 30 | 120 | 172 | 20,640 | 5,160 | 1,720 |
| Sub-totals: | | | 1,562 | 112,464 | 28,116 | 9,372 |
| 400 | 6 | 60 | 172 | 10,320 | 1,032 | 344 |
| | 9 | 90 | 174 | 15,660 | 1,566 | 522 |
| | 12 | 120 | 174 | 20,880 | 2,088 | 696 |
| | 15 | 150 | 174 | 26,100 | 2,610 | 870 |
| | 18 | 180 | 174 | 31,320 | 3,132 | 1,044 |
| | 21 | 210 | 174 | 36,540 | 3,654 | 1,218 |
| | 24 | 240 | 174 | 41,760 | 4,176 | 1,392 |
| | 27 | 270 | 174 | 46,980 | 4,698 | 1,566 |
| | 30 | 300 | 172 | 51,600 | 5,160 | 1,720 |
| Sub-totals: | | | 1,562 | 281,160 | 28,116 | 9,372 |
| 200 | 6 | 120 | 172 | 20,640 | 1,032 | 344 |
| | 9 | 180 | 174 | 31,320 | 1,566 | 522 |
| | 12 | 240 | 174 | 41,760 | 2,088 | 696 |
| | 15 | 300 | 174 | 52,200 | 2,610 | 870 |
| | 18 | 360 | 174 | 62,640 | 3,132 | 1,044 |
| | 21 | 420 | 174 | 73,080 | 3,654 | 1,218 |
| | 24 | 480 | 174 | 83,520 | 4,176 | 1,392 |
| | 27 | 540 | 174 | 93,960 | 4,698 | 1,566 |
| | 30 | 600 | 172 | 103,200 | 5,160 | 1,720 |
| Sub-totals: | | | 1,562 | 562,320 | 28,116 | 9,372 |

TABLE 52. RECORD/BLOCK DISTRIBUTION, LARGE DATABASE, CONFIGURATION 1.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 2 | 4 | 344 | 1,376 | 688 | 688 |
| | 3 | 6 | 348 | 2,088 | 1,044 | 1,044 |
| | 4 | 8 | 348 | 2,784 | 1,392 | 1,392 |
| | 5 | 10 | 348 | 3,480 | 1,740 | 1,740 |
| | 6 | 12 | 348 | 4,176 | 2,088 | 2,088 |
| | 7 | 14 | 348 | 4,872 | 2,436 | 2,436 |
| | 8 | 16 | 348 | 5,568 | 2,784 | 2,784 |
| | 9 | 18 | 348 | 6,264 | 3,132 | 3,132 |
| | 10 | 20 | 344 | 6,880 | 3,440 | 3,440 |
| Sub-totals: | | | 3,124 | 37,488 | 18,744 | 18,744 |
| 1000 | 2 | 8 | 344 | 2,752 | 688 | 688 |
| | 3 | 12 | 348 | 4,176 | 1,044 | 1,044 |
| | 4 | 16 | 348 | 5,568 | 1,392 | 1,392 |
| | 5 | 20 | 348 | 6,960 | 1,740 | 1,740 |
| | 6 | 24 | 348 | 8,352 | 2,088 | 2,088 |
| | 7 | 28 | 348 | 9,744 | 2,436 | 2,436 |
| | 8 | 32 | 348 | 11,136 | 2,784 | 2,784 |
| | 9 | 36 | 348 | 12,528 | 3,132 | 3,132 |
| | 10 | 40 | 344 | 13,760 | 3,440 | 3,440 |
| Sub-totals: | | | 3,124 | 74,976 | 18,744 | 18,744 |
| 400 | 2 | 20 | 344 | 6,880 | 688 | 688 |
| | 3 | 30 | 348 | 10,440 | 1,044 | 1,044 |
| | 4 | 40 | 348 | 13,920 | 1,392 | 1,392 |
| | 5 | 50 | 348 | 17,400 | 1,740 | 1,740 |
| | 6 | 60 | 348 | 20,880 | 2,088 | 2,088 |
| | 7 | 70 | 348 | 24,360 | 2,436 | 2,436 |
| | 8 | 80 | 348 | 27,840 | 2,784 | 2,784 |
| | 9 | 90 | 348 | 31,320 | 3,132 | 3,132 |
| | 10 | 100 | 344 | 34,400 | 3,440 | 3,440 |
| Sub-Totals: | | | 3,124 | 187,440 | 18,744 | 18,744 |
| 200 | 2 | 40 | 344 | 13,760 | 688 | 688 |
| | 3 | 60 | 348 | 20,880 | 1,044 | 1,044 |
| | 4 | 80 | 348 | 27,840 | 1,392 | 1,392 |
| | 5 | 100 | 348 | 34,800 | 1,740 | 1,740 |
| | 6 | 120 | 348 | 41,760 | 2,088 | 2,088 |
| | 7 | 140 | 348 | 48,720 | 2,436 | 2,436 |
| | 8 | 160 | 348 | 55,680 | 2,784 | 2,784 |
| | 9 | 180 | 348 | 62,640 | 3,132 | 3,132 |
| | 10 | 200 | 344 | 68,800 | 3,440 | 3,440 |
| Sub-totals: | | | 3,124 | 374,880 | 18,744 | 18,744 |

TABLE 53. RECORD/BLOCK DISTRIBUTION.
LARGE DATABASE, CONFIGURATION 2.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 2 | 4 | 344 | 1,376 | 688 | 344 |
| | 3 | 6 | 348 | 2,088 | 1,044 | 522 |
| | 4 | 8 | 348 | 2,784 | 1,392 | 696 |
| | 5 | 10 | 348 | 3,480 | 1,740 | 870 |
| | 6 | 12 | 348 | 4,176 | 2,088 | 1,044 |
| | 7 | 14 | 348 | 4,872 | 2,436 | 1,218 |
| | 8 | 16 | 348 | 5,568 | 2,784 | 1,392 |
| | 9 | 18 | 348 | 6,264 | 3,132 | 1,566 |
| | 10 | 20 | 344 | 6,880 | 3,440 | 1,720 |
| Sub-totals: | | | 3,124 | 37,488 | 18,744 | 9,372 |
| 1000 | 2 | 8 | 344 | 2,752 | 688 | 344 |
| | 3 | 12 | 348 | 4,176 | 1,044 | 522 |
| | 4 | 16 | 348 | 5,568 | 1,392 | 696 |
| | 5 | 20 | 348 | 6,960 | 1,740 | 870 |
| | 6 | 24 | 348 | 8,352 | 2,088 | 1,044 |
| | 7 | 28 | 348 | 9,744 | 2,436 | 1,218 |
| | 8 | 32 | 348 | 11,136 | 2,784 | 1,392 |
| | 9 | 36 | 348 | 12,528 | 3,132 | 1,566 |
| | 10 | 40 | 344 | 13,760 | 3,440 | 1,720 |
| Sub-totals: | | | 3,124 | 74,976 | 18,744 | 9,372 |
| 400 | 2 | 20 | 344 | 6,880 | 688 | 344 |
| | 3 | 30 | 348 | 10,440 | 1,044 | 522 |
| | 4 | 40 | 348 | 13,920 | 1,392 | 696 |
| | 5 | 50 | 348 | 17,400 | 1,740 | 870 |
| | 6 | 60 | 348 | 20,880 | 2,088 | 1,044 |
| | 7 | 70 | 348 | 24,360 | 2,436 | 1,218 |
| | 8 | 80 | 348 | 27,840 | 2,784 | 1,392 |
| | 9 | 90 | 348 | 31,320 | 3,132 | 1,566 |
| | 10 | 100 | 344 | 34,400 | 3,440 | 1,720 |
| Sub-totals: | | | 3,124 | 187,440 | 18,744 | 9,372 |
| 200 | 2 | 40 | 344 | 13,760 | 688 | 344 |
| | 3 | 60 | 348 | 20,880 | 1,044 | 522 |
| | 4 | 80 | 348 | 27,840 | 1,392 | 696 |
| | 5 | 100 | 348 | 34,800 | 1,740 | 870 |
| | 6 | 120 | 348 | 41,760 | 2,088 | 1,044 |
| | 7 | 140 | 348 | 48,720 | 2,436 | 1,218 |
| | 8 | 160 | 348 | 55,680 | 2,784 | 1,392 |
| | 9 | 180 | 348 | 62,640 | 3,132 | 1,566 |
| | 10 | 200 | 344 | 68,800 | 3,440 | 1,720 |
| Sub-totals: | | | 3,124 | 374,880 | 18,744 | 9,372 |

TABLE 54a. RECORD/BLOCK DISTRIBUTION. LARGE DATABASE, CONFIGURATION 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 2 | 4 | 344 | 1,376 | 688 | |
| | 3 | 6 | 348 | 2,088 | 1,044 | |
| | 4 | 8 | 348 | 2,784 | 1,392 | (See |
| | 5 | 10 | 348 | 3,480 | 1,740 | below) |
| | 6 | 12 | 348 | 4,176 | 2,088 | |
| | 7 | 14 | 348 | 4,872 | 2,436 | |
| | 8 | 16 | 348 | 5,568 | 2,784 | |
| | 9 | 18 | 348 | 6,264 | 3,132 | |
| | 10 | 20 | 344 | 6,880 | 3,440 | |
| Sub-totals: | | | 3,124 | 37,488 | 18,744 | |

| Number of Blocks per Cluster | Backend #1 | | Backend #2 | | Backend #3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 232 | 464 | 228 | 456 | 228 | 456 |
| 3 | 348 | 696 | 348 | 696 | 348 | 696 |
| 4 | 464 | 928 | 464 | 928 | 464 | 928 |
| 5 | 580 | 1,160 | 580 | 1,160 | 580 | 1,160 |
| 6 | 696 | 1,392 | 696 | 1,392 | 696 | 1,392 |
| 7 | 812 | 1,624 | 812 | 1,624 | 812 | 1,624 |
| 8 | 928 | 1,856 | 928 | 1,856 | 928 | 1,856 |
| 9 | 1,044 | 2,088 | 1,044 | 2,088 | 1,044 | 2,088 |
| 10 | 1,144 | 2,288 | 1,148 | 2,296 | 1,148 | 2,296 |
| Sub-totals: | 6,248 | 12,496 | 6,248 | 12,496 | 6,248 | 12,496 |

TABLE 54b. RECORD/BLOCK DISTRIBUTION, LARGE DATABASE, CONFIGURATION 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 1000 | 2 | 8 | 344 | 2,752 | 688 | |
| | 3 | 12 | 348 | 4,176 | 1,044 | |
| | 4 | 16 | 348 | 5,568 | 1,392 | (See |
| | 5 | 20 | 348 | 6,960 | 1,740 | below) |
| | 6 | 24 | 348 | 8,352 | 2,088 | |
| | 7 | 28 | 348 | 9,744 | 2,436 | |
| | 8 | 32 | 348 | 11,136 | 2,784 | |
| | 9 | 36 | 348 | 12,528 | 3,132 | |
| | 10 | 40 | 344 | 13,760 | 3,440 | |
| Sub-totals: | | | 3,124 | 74,976 | 18,744 | |

| Number of Blocks per Cluster | Backend #1 | | Backend #2 | | Backend #3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 232 | 928 | 228 | 912 | 228 | 912 |
| 3 | 348 | 1,392 | 348 | 1,392 | 348 | 1,392 |
| 4 | 464 | 1,856 | 464 | 1,856 | 464 | 1,856 |
| 5 | 580 | 2,320 | 580 | 2,320 | 580 | 2,320 |
| 6 | 696 | 2,784 | 696 | 2,784 | 696 | 2,784 |
| 7 | 812 | 3,248 | 812 | 3,248 | 812 | 3,248 |
| 8 | 928 | 3,712 | 928 | 3,712 | 928 | 3,712 |
| 9 | 1,044 | 4,176 | 1,044 | 4,176 | 1,044 | 4,176 |
| 10 | 1,144 | 4,576 | 1,148 | 4,592 | 1,148 | 4,592 |
| Sub-totals: | 6,248 | 24,992 | 6,248 | 24,992 | 6,248 | 24,992 |

TABLE 54c. RECORD/BLOCK DISTRIBUTION, LARGE DATABASE, CONFIGURATION 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 400 | 2 | 20 | 344 | 6,880 | 688 | |
| | 3 | 30 | 348 | 10,440 | 1,044 | |
| | 4 | 40 | 348 | 13,920 | 1,392 | (See |
| | 5 | 50 | 348 | 17,400 | 1,740 | below) |
| | 6 | 60 | 348 | 20,880 | 2,088 | |
| | 7 | 70 | 348 | 24,360 | 2,436 | |
| | 8 | 80 | 348 | 27,840 | 2,784 | |
| | 9 | 90 | 348 | 31,320 | 3,132 | |
| | 10 | 100 | 344 | 34,400 | 3,440 | |
| Sub-totals: | | | 3,124 | 187,440 | 18,744 | |

| Number of Blocks per Cluster | Backend #1 | | Backend #2 | | Backend #3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 232 | 2,320 | 228 | 2,280 | 228 | 2,280 |
| 3 | 348 | 3,480 | 348 | 3,480 | 348 | 3,480 |
| 4 | 464 | 4,640 | 464 | 4,640 | 464 | 4,640 |
| 5 | 580 | 5,800 | 580 | 5,800 | 580 | 5,800 |
| 6 | 696 | 6,960 | 696 | 6,960 | 696 | 6,960 |
| 7 | 812 | 8,120 | 812 | 8,120 | 812 | 8,120 |
| 8 | 928 | 9,280 | 928 | 9,280 | 928 | 9,280 |
| 9 | 1,044 | 10,440 | 1,044 | 10,440 | 1,044 | 10,440 |
| 10 | 1,144 | 11,440 | 1,148 | 11,480 | 1,148 | 11,480 |
| Sub-totals: | 6,248 | 62,480 | 6,248 | 62,480 | 6,248 | 62,480 |

TABLE 54d. RECORD/BLOCK DISTRIBUTION, LARGE DATABASE, CONFIGURATION 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 200 | 2 | 40 | 344 | 13,760 | 688 | |
| | 3 | 60 | 348 | 20,880 | 1,044 | |
| | 4 | 80 | 348 | 27,840 | 1,392 | (See |
| | 5 | 100 | 348 | 34,800 | 1,740 | below) |
| | 6 | 120 | 348 | 41,760 | 2,088 | |
| | 7 | 140 | 348 | 48,720 | 2,436 | |
| | 8 | 160 | 348 | 55,680 | 2,784 | |
| | 9 | 180 | 348 | 62,640 | 3,132 | |
| | 10 | 200 | 344 | 68,800 | 3,440 | |
| Sub-totals: | | | 3,124 | 374,880 | 18,744 | |

| Number of Blocks per Cluster | Backend #1 | | Backend #2 | | Backend #3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 232 | 4,640 | 228 | 4,560 | 228 | 4,560 |
| 3 | 348 | 6,960 | 348 | 6,960 | 348 | 6,960 |
| 4 | 464 | 9,280 | 464 | 9,280 | 464 | 9,280 |
| 5 | 580 | 11,600 | 580 | 11,600 | 580 | 11,600 |
| 6 | 696 | 13,920 | 696 | 13,920 | 696 | 13,920 |
| 7 | 812 | 16,240 | 812 | 16,240 | 812 | 16,240 |
| 8 | 928 | 18,560 | 928 | 18,560 | 928 | 18,560 |
| 9 | 1,044 | 20,880 | 1,044 | 20,880 | 1,044 | 20,880 |
| 10 | 1,144 | 22,880 | 1,148 | 22,960 | 1,148 | 22,960 |
| Sub-totals: | 6,248 | 124,960 | 6,248 | 124,960 | 6,248 | 124,960 |

TABLE 55. RECORD BLOCK DISTRIBUTION, LARGE DATABASE, CONFIGURATION 4.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 4 | 8 | 344 | 2,752 | 1,376 | 688 |
| | 6 | 12 | 348 | 4,176 | 2,088 | 1,044 |
| | 8 | 16 | 348 | 5,568 | 2,784 | 1,392 |
| | 10 | 20 | 348 | 6,960 | 3,480 | 1,740 |
| | 12 | 24 | 348 | 8,352 | 4,176 | 2,088 |
| | 14 | 28 | 348 | 9,744 | 4,872 | 2,436 |
| | 16 | 32 | 348 | 11,136 | 5,568 | 2,784 |
| | 18 | 36 | 348 | 12,528 | 6,264 | 3,132 |
| | 20 | 40 | 344 | 13,760 | 6,880 | 3,440 |
| Sub-Totals: | | | 3,124 | 74,976 | 37,488 | 18,744 |
| 1000 | 4 | 16 | 344 | 5,504 | 1,376 | 688 |
| | 6 | 24 | 348 | 8,352 | 2,088 | 1,044 |
| | 8 | 32 | 348 | 11,136 | 2,784 | 1,392 |
| | 10 | 40 | 348 | 13,920 | 3,480 | 1,740 |
| | 12 | 48 | 348 | 16,704 | 4,176 | 2,088 |
| | 14 | 56 | 348 | 19,488 | 4,872 | 2,436 |
| | 16 | 64 | 348 | 22,272 | 5,568 | 2,784 |
| | 18 | 72 | 348 | 25,056 | 6,264 | 3,132 |
| | 20 | 80 | 344 | 27,520 | 6,880 | 3,440 |
| Sub-totals: | | | 3,124 | 149,952 | 37,488 | 18,744 |
| 400 | 4 | 40 | 344 | 13,760 | 1,376 | 688 |
| | 6 | 60 | 348 | 20,880 | 2,088 | 1,044 |
| | 8 | 80 | 348 | 27,840 | 2,784 | 1,392 |
| | 10 | 100 | 348 | 34,800 | 3,480 | 1,740 |
| | 12 | 120 | 348 | 41,760 | 4,176 | 2,088 |
| | 14 | 140 | 348 | 48,720 | 4,872 | 2,436 |
| | 16 | 160 | 348 | 55,680 | 5,568 | 2,784 |
| | 18 | 180 | 348 | 62,640 | 6,264 | 3,132 |
| | 20 | 200 | 344 | 68,800 | 6,880 | 3,440 |
| Sub-totals: | | | 3,124 | 374,880 | 37,488 | 18,744 |
| 200 | 4 | 80 | 344 | 27,520 | 1,376 | 688 |
| | 6 | 120 | 348 | 41,760 | 2,088 | 1,044 |
| | 8 | 160 | 348 | 55,680 | 2,784 | 1,392 |
| | 10 | 200 | 348 | 69,600 | 3,480 | 1,740 |
| | 12 | 240 | 348 | 83,520 | 4,176 | 2,088 |
| | 14 | 280 | 348 | 97,440 | 4,872 | 2,436 |
| | 16 | 320 | 348 | 111,360 | 5,568 | 2,784 |
| | 18 | 360 | 348 | 125,280 | 6,264 | 3,132 |
| | 20 | 400 | 344 | 137,600 | 6,880 | 3,440 |
| Sub-totals: | | | 3,124 | 749,760 | 37,488 | 18,744 |

TABLE 56. RECORD BLOCK DISTRIBUTION, LARGE DATABASE. CONFIGURATION 5.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 6 | 12 | 344 | 4,128 | 2,064 | 688 |
| | 9 | 18 | 348 | 6,264 | 3,132 | 1,044 |
| | 12 | 24 | 348 | 8,352 | 4,176 | 1,392 |
| | 15 | 30 | 348 | 10,440 | 5,220 | 1,740 |
| | 18 | 36 | 348 | 12,528 | 6,264 | 2,088 |
| | 21 | 42 | 348 | 14,616 | 7,308 | 2,436 |
| | 24 | 48 | 348 | 16,704 | 8,352 | 2,784 |
| | 27 | 54 | 348 | 18,792 | 9,396 | 3,132 |
| | 30 | 60 | 344 | 20,640 | 10,320 | 3,440 |
| Sub-Totals: | | | 3,124 | 112,464 | 56,232 | 18,744 |
| 1000 | 6 | 24 | 344 | 8,256 | 2,064 | 688 |
| | 9 | 36 | 348 | 12,528 | 3,132 | 1,044 |
| | 12 | 48 | 348 | 16,704 | 4,176 | 1,392 |
| | 15 | 60 | 348 | 20,880 | 5,220 | 1,740 |
| | 18 | 72 | 348 | 25,056 | 6,264 | 2,088 |
| | 21 | 84 | 348 | 29,232 | 7,308 | 2,436 |
| | 24 | 96 | 348 | 33,408 | 8,352 | 2,784 |
| | 27 | 108 | 348 | 37,584 | 9,396 | 3,132 |
| | 30 | 120 | 344 | 41,280 | 10,320 | 3,440 |
| Sub-Totals: | | | 3,124 | 224,928 | 56,232 | 18,744 |
| 400 | 6 | 60 | 344 | 20,640 | 2,064 | 688 |
| | 9 | 90 | 348 | 31,320 | 3,132 | 1,044 |
| | 12 | 120 | 348 | 41,760 | 4,176 | 1,392 |
| | 15 | 150 | 348 | 52,200 | 5,220 | 1,740 |
| | 18 | 180 | 348 | 62,640 | 6,264 | 2,088 |
| | 21 | 210 | 348 | 73,080 | 7,308 | 2,436 |
| | 24 | 240 | 348 | 83,520 | 8,352 | 2,784 |
| | 27 | 270 | 348 | 93,960 | 9,396 | 3,132 |
| | 30 | 300 | 344 | 103,200 | 10,320 | 3,440 |
| Sub-Totals: | | | 3,124 | 562,320 | 56,232 | 18,744 |
| 200 | 6 | 120 | 344 | 41,280 | 2,064 | 688 |
| | 9 | 180 | 348 | 62,640 | 3,132 | 1,044 |
| | 12 | 240 | 348 | 83,520 | 4,176 | 1,392 |
| | 15 | 300 | 348 | 104,440 | 5,220 | 1,740 |
| | 18 | 360 | 348 | 125,280 | 6,264 | 2,088 |
| | 21 | 420 | 348 | 146,160 | 7,308 | 2,436 |
| | 24 | 480 | 348 | 167,040 | 8,352 | 2,784 |
| | 27 | 540 | 348 | 187,920 | 9,396 | 3,132 |
| | 30 | 600 | 344 | 206,400 | 10,320 | 3,440 |
| Sub-Totals: | | | 3,124 | 1,124,640 | 56,232 | 18,744 |

The MBDS requires that all attributes in a record be the same size. Variable-length fields within a record **are not** supported. Since the four record classes we have chosen are all divisible by 10, we set the attribute size to 10-bytes per attribute. Table 57 shows the number of 10-byte attributes corresponding to each record class.

TABLE 57. NUMBER OF 10-BYTE ATTRIBUTES PER RECORD CLASS.

| Record Size in Bytes | Number of Attributes |
|---|---|
| 2000 | 200 |
| 1000 | 100 |
| 400 | 40 |
| 200 | 20 |

The technique we apply for defining the record templates, descriptor types, and descriptor ranges is a variation of the scheme proposed in [Ref. 6: pp. 72-76, Ref. 10: pp. 9-12, and Ref. 12: pp. 11-20]. We specify the record templates for each record class in Table 58. For the four record templates listed, the TEMPLATE, INT2001, INT1001, INT401, INT201, INT2002, INT1002, INT402, and INT202 attributes are directory attributes, while the remaining attributes of each template are non-directory attributes. We also note that TEMPLATE is a type-B attribute, whereas the INTxx1 and INTxx2 attributes are of type-A.

Next, we must describe the range of values for each of the record attributes listed in Table 58. We begin by considering the descriptor types, (i.e., type-A, type-B, or type-C), and the descriptor ranges for the directory attributes TEMPLATE, INTxx1, and INTxx2. The nine directory attributes and their corresponding descriptor identifiers are listed in Table 59.

The TEMPLATE attribute is used to correlate each record with its corresponding record template. This attribute may take on the four values listed in Table 60, corresponding to the four record classes. Note in Table 60 that we use the notation Di-j to label descriptor identifiers. This represents the jth descriptor for the ith directory attribute.

The range of values for the INTxx1 attributes for each record template are a function of the individual record-class, (2000, 1000, 400, or 200-bytes), the

TABLE 58a. RECORD TEMPLATE FOR 2000-BYTE RECORD CLASS.

| Attribute Number | Attribute Name | Attribute Type |
|---|---|---|
| 1 | TEMPLATE | string |
| 2 | INT2001 | integer |
| 3 | INT2002 | integer |
| 4 | MULTIPLE | string |
| 5 | STRING001 | string |
| 6 | STRING002 | string |
| . | . | . |
| . | . | . |
| . | . | . |
| 199 | STRING195 | string |
| 200 | STRING196 | string |

TABLE 58b. RECORD TEMPLATE FOR 1000-BYTE RECORD CLASS.

| Attribute Number | Attribute Name | Attribute Type |
|---|---|---|
| 1 | TEMPLATE | string |
| 2 | INT1001 | integer |
| 3 | INT1002 | integer |
| 4 | MULTIPLE | string |
| 5 | STRING001 | string |
| 6 | STRING002 | string |
| . | . | . |
| . | . | . |
| . | . | . |
| 99 | STRING095 | string |
| 100 | STRING096 | string |

TABLE 58c. RECORD TEMPLATE FOR 400-BYTE RECORD CLASS.

| Attribute Number | Attribute Name | Attribute Type |
|---|---|---|
| 1 | TEMPLATE | string |
| 2 | INT401 | integer |
| 3 | INT402 | integer |
| 4 | MULTIPLE | string |
| 5 | STRING001 | string |
| 6 | STRING002 | string |
| . | . | . |
| . | . | . |
| . | . | . |
| 39 | STRING035 | string |
| 40 | STRING036 | string |

TABLE 58d. RECORD TEMPLATE FOR 200-BYTE RECORD CLASS.

| Attribute Number | Attribute Name | Attribute Type |
|---|---|---|
| 1 | TEMPLATE | string |
| 2 | INT201 | integer |
| 3 | INT202 | integer |
| 4 | MULTIPLE | string |
| 5 | STRING001 | string |
| 6 | STRING002 | string |
| . | . | . |
| . | . | . |
| . | . | . |
| 19 | STRING015 | string |
| 20 | STRING016 | string |

TABLE 59. THE DIRECTORY ATTRIBUTES AND THEIR DESCRIPTORS.

| Attribute Number | Attribute Name | Descriptor Identifier | Descriptor Type |
|---|---|---|---|
| 1 | TEMPLATE | D1-j | B |
| 2 | INT2001 | D2-j | A |
| 3 | INT1001 | D3-j | A |
| 4 | INT401 | D4-j | A |
| 5 | INT201 | D5-j | A |
| 6 | INT2002 | D6-j | A |
| 7 | INT1002 | D7-j | A |
| 8 | INT402 | D8-j | A |
| 9 | INT202 | D9-j | A |

TABLE 60. TEMPLATE ATTRIBUTE VALUES.

| TEMPLATE Value | Descriptor Identifier |
|---|---|
| TEMP2000 | D1-1 |
| TEMP1000 | D1-2 |
| TEMP400 | D1-3 |
| TEMP200 | D1-4 |

database-size category, (small, medium, or large), and the test configuration, (1, 2, 3, 4, or 5). Table 32 shows that the total database size remains constant for test configurations 1, 2, and 3. Therefore, for a given database-size category we require three sets of descriptors, corresponding to databases with a total of N, 2N, and 3N Mbytes per database. This means that a total of nine databases are required for the test-database set to test MBDS with a maximum of three backends. In the discussion to follow, we will refer to these nine databases by the acronyms DB1 to DB9, as described in Table 61.

We use database DB1, which is used for configurations 1, 2, and 3 of Table 33, to develop the value ranges for the remaining record attributes. The entries for configuration 1 of Table 33 specify 9,372 2000-byte records, 18,744 1000-byte records, 46,860 400-byte records, and 93,720 200-byte records. We use nine type-A descriptors to classify the values for the INTxx1 attributes, corresponding

TABLE 61. LIST OF TEST DATABASE ACRONYMS.

| Test Database Acronym | Database Size Category | Database Size in Mbytes |
|---|---|---|
| DB1 | Small | N = 74.976 |
| DB2 | Small | 2N = 149.952 |
| DB3 | Small | 3N = 224.928 |
| DB4 | Medium | N = 149.952 |
| DB5 | Medium | 2N = 299.904 |
| DB6 | Medium | 3N = 449.856 |
| DB7 | Large | N = 299.904 |
| DB8 | Large | 2N = 599.808 |
| DB9 | Large | 3N = 899.712 |

to the nine cluster categories of Table 37. For the DB1 database, column 5 of Tables 41/42 (Total Number of Records) shows the pertinent values to use for these nine descriptors. The range of values for the nine descriptors for each of the INT2001, INT1001, INT401, and INT201 attributes are listed in Table 62.

The third directory attribute, INTxx2, enables us to distribute the records in each of the nine cluster categories identified by the INTxx1 attributes into subsets. Referring to column 4 of Tables 41/42, we see that the easiest way to

TABLE 62. THE INTxx1 ATTRIBUTES AND DESCRIPTORS.

| Directory Attribute | Descriptor Identifier | Range of Values | Number of Records |
|---|---|---|---|
| INT2001 | D2-1 | [1;344] | 344 |
| | D2-2 | [345;866] | 522 |
| | D2-3 | [867;1,562] | 696 |
| | D2-4 | [1,563;2,432] | 870 |
| | D2-5 | [2,433;3,476] | 1,044 |
| | D2-6 | [3,477;4,694] | 1,218 |
| | D2-7 | [4,695;6,086] | 1,392 |
| | D2-8 | [6,087;7,652] | 1,566 |
| | D2-9 | [7,653;9,372] | 1,720 |
| INT1001 | D3-1 | [1;688] | 688 |
| | D3-2 | [689;1,732] | 1,044 |
| | D3-3 | [1,733;3,124] | 1,392 |
| | D3-4 | [3,125;4,864] | 1,740 |
| | D3-5 | [4,865;6,952] | 2,088 |
| | D3-6 | [6,953;9,388] | 2,436 |
| | D3-7 | [9,389;12,172] | 2,784 |
| | D3-8 | [12,173;15.304] | 3,132 |
| | D3-9 | [15,304;18.744] | 3,440 |

TABLE 62. THE INTxx1 ATTRIBUTES AND DESCRIPTORS. (cont'd).

| Directory Attribute | Descriptor Identifier | Range of Values | Number of Records |
|---|---|---|---|
| INT401 | D4-1 | [1;1,720] | 1,720 |
| | D4-2 | [1,721;4,330] | 2,610 |
| | D4-3 | [4,331;7,810] | 3,480 |
| | D4-4 | [7,811;12,160] | 4,350 |
| | D4-5 | [12,161;17,380] | 5,220 |
| | D4-6 | [17,381;23,470] | 6,090 |
| | D4-7 | [23,471;30,430] | 6,960 |
| | D4-8 | [30,431;38,260] | 7,830 |
| | D4-9 | [38,261;46,860] | 8,600 |
| INT201 | D5-1 | [1;3,440] | 3,440 |
| | D5-2 | [3,441;8,660] | 5,220 |
| | D5-3 | [8,661;15,620] | 6,960 |
| | D5-4 | [15,621;24,320] | 8,700 |
| | D5-5 | [24,321;34,760] | 10,440 |
| | D5-6 | [34,761;46,940] | 12,180 |
| | D5-7 | [46,941;60,860] | 13,920 |
| | D5-8 | [60,861;76,520] | 15,660 |
| | D5-9 | [76,521;93,720] | 17,200 |

subdivide each cluster category is into individual clusters. We use 781 type-A descriptors to classify the values for each of the four INTxx2 attributes for the DB1 database. If we consider attribute INT2002 for the 2000-byte record class, we see that we have 86 clusters with 4 records per cluster for a total of 344 records. Therefore, we use 86 descriptors, one per cluster, as shown in Table 63 for the first cluster category, which is identified by the INT2001 descriptor, D2-1.

The INTxx2 attribute-value ranges are calculated via the relationship [w + xy - (x-1); w + xy], which is described in Figure 18. The lower bound of the

range is represented by the term (w + xy - (x-1)), while the second term, w + xy, represents the upper bound. Applying this relationship for the first cluster of the 2000-byte record class for the DB1 database, we use w = 0, x = 4, and y = {1,...,86}. Therefore, the range of values for INT2002 becomes [1;4], [5;8], [9;12], ..., [341;344]. For the second cluster, we have w = 344, x = 6, and y = {1,...,87}, to derive the ranges [345;350], [351;356], ..., [861;866]. Continuing in this manner, we derive the entries shown in Table 63 for the INT2002 range of values.

We do not present tables for the corresponding values for the INT1002, INT402, and INT202 attributes, since the procedure for deriving these values is identical to that shown for Table 63. Note, however, that the INT1002 descriptor ID's range from D7-1 to D7-781: the INT402 descriptor ID's range from D8-1 to D8-781; and the INT202 descriptor ID's range form D9-1 to D9-781.

The MULTIPLE attribute is a character string which enables us to easily increase the number of records within each cluster [Ref. 6: p. 72]. Recall that this is required when we need to double or triple the database size to test configurations 4 and 5, as shown in Tables 45 and 46. For configurations 1, 2, and 3, which use the DB1 database, MULTIPLE is set to 'One'. To double the database size for configuration 4, each (INTxx1, INTxx2) pair must match up with MULTIPLE attribute values of 'One' and 'Two'. To triple the database size for configuration 5, each (INTxx1, INTxx2) pair must match up with MULTIPLE attribute values of 'One', 'Two', and 'Three'. This relationship is shown in Table 64.

Finally, the STRINGxxx attributes are used as filler fields, and are all set to the character-string value Xxxxxxxxx. Note that this represents a nine-character string, requiring nine-bytes of storage, whereas the allocated attribute size is ten-bytes. The reason that only nine characters are used is that the C language compiler inserts a null character, (i.e., a backslash-zero), to mark the end of each character string [Ref. 20: pp. 35-36]. Therefore, although we use ten-byte attributes, we only have nine usable bytes for our character-string values. The STRINGxxx attributes are also used to allow flexibility in retrieving portions of the database. For example, in the test-transaction mix we present in Chapter

TABLE 63. INT2002 ATTRIBUTE VALUE RANGES.

| INT2001 Descriptor Identifier | INT2001 Range of Values | INT2002 Descriptor Identifier | INT2002 Range of Values |
|---|---|---|---|
| D2-1 | [1;344] | D6-1 | [1;4] |
| | | D6-2 | [5;8] |
| | | ... | ... |
| | | D6-86 | [341;344] |
| D2-2 | [345;866] | D6-87 | [345;350] |
| | | D6-88 | [351;356] |
| | | ... | ... |
| | | D6-173 | [861;866] |
| D2-3 | [867;1,562] | D6-174 | [867;874] |
| | | D6-175 | [875;882] |
| | | ... | ... |
| | | D6-260 | [1,555;1,562] |
| D2-4 | [1,563;2,432] | D6-261 | [1,563;1,572] |
| | | D6-262 | [1,573;1,582] |
| | | ... | ... |
| | | D6-347 | [2,423;2,432] |
| D2-5 | [2,433;3,476] | D6-348 | [2,433;2,444] |
| | | D6-349 | [2,445;2,456] |
| | | ... | ... |
| | | D6-434 | [3,465;3,476] |
| D2-6 | [3,477;4,694] | D6-435 | [3,477;3,490] |
| | | D6-436 | [3,491;3,504] |
| | | ... | ... |
| | | D6-521 | [4,681;4,694] |
| D2-7 | [4,695;6,086] | D6-522 | [4,695;4,710] |
| | | D6-523 | [4,711;4,726] |
| | | ... | ... |
| | | D6-608 | [6,071;6,086] |
| D2-8 | [6,087;7,652] | D6-609 | [6,087;6,104] |
| | | D6-610 | [6,105;6,122] |
| | | ... | ... |
| | | D6-695 | [7,635;7,652] |
| D2-9 | [7,653;9,372] | D6-696 | [7,653;7,672] |
| | | D6-697 | [7,673;7,692] |
| | | ... | ... |
| | | D6-781 | [9,353;9,372] |

[lower-bound; upper-bound] = [w + xy - (x - 1); w + xy]

where:

w = sum of records from all previous clusters.

=> Initially, w = 0;

=> At end of each cluster category, before advancing to the next INTxx1 descriptor, reset w.

=> w = w + xy,
where y is the max value for this INTxx1 descriptor.

x = Number of record per cluster from Tables V-11/V-12.

{4, 6, 8, 10, 12, 14, 16, 18, 20} for 2000-byte records.

{8, 12, 16, 20, 24, 28, 32, 36, 40} for 1000-byte records.

{20, 30, 40, 50, 60, 70, 80, 90, 100} for 400-byte records.

{40, 60, 80, 100, 120, 140, 160, 180, 200} for 200-byte records.

y = {1, ..., z}

z = {{86,87}, {172, 174}, {344, 348}}

=> z = {86, 87} for small database, (N/4).

=> z = {172, 174} for medium database, (N/2).

=> z = {344, 348} for large database, (N).

Figure 18. INTxx2 Attribute-Value Range Relationship.

TABLE 64. USE OF MULTIPLE FOR DATABASE DB3 (3N MBYTES).

| TEMPLATE | INT2001 | INT2002 | MULTIPLE |
|---|---|---|---|
| TEMP2000 | 1 | 1 | One |
| | 2 | 2 | One |
| | . . . | . . . | . . |
| | 9,372 | 9,372 | One |
| | 1 | 1 | Two |
| | 2 | 2 | Two |
| | . . . | . . . | . . . |
| | 9,372 | 9,372 | Two |
| | 1 | 1 | Three |
| | 2 | 2 | Three |
| | . . . | . . . | . . . |
| | 9,372 | 9,372 | Three |

VI, we use UPDATE operations to update certain STRINGxxx attributes to values such as OneEighth, One-Qtr, and One-Half. We then use RETRIEVE requests which key on the applicable STRINGxxx fields to retrieve 1/8, 1/4, and 1/2 of the database, respectively.

We have now described all of the attributes for the record templates of Table 58. The general layout of the 2000-byte record file for the DB1 database is shown in Table 65.

TABLE 65. LAYOUT OF THE 2000-BYTE RECORD FILE FOR DB1.

| TEMPLATE | INT2001 | INT2002 | MULTIPLE | STRING001 | . . . | STRING196 |
|---|---|---|---|---|---|---|
| TEMP2000 | 1 | 1 | One | Xxxxxxxxx | . . . | Xxxxxxxxx |
| TEMP2000 | 2 | 2 | One | Xxxxxxxxx | . . . | Xxxxxxxxx |
| TEMP2000 | 3 | 3 | One | Xxxxxxxxx | . . . | Xxxxxxxxx |
| TEMP2000 | 4 | 4 | One | Xxxxxxxxx | . . . | Xxxxxxxxx |
| . . . | . . . | . . . | . . . | . . . | . . . | . . . |
| TEMP2000 | 9,371 | 9,371 | One | Xxxxxxxxx | . . . | Xxxxxxxxx |
| TEMP2000 | 9,372 | 9,372 | One | Xxxxxxxxx | . . . | Xxxxxxxxx |

In this section we have specified the record templates for each of the four record classes for the test database, and we have developed the descriptors for the 2000-byte record file for the DB1 database. The development of the descriptors for the rest of the DB1 files is straightforward. and follows the steps presented above for the 2000-byte record case exactly.

The descriptors for the DB2 and DB3 databases are also developed as presented above. The only major change is that the number of records per cluster doubles (for DB2) or triples (for DB3). Therefore, the corresponding range of values for the INTxx1 and INTxx2 descriptors must double or triple from those shown in Tables 62 and 63.

For the DB4-DB6 databases, there are 1,562 INTxx2 descriptors for each record template, since the number of clusters doubles for the medium-size database. Similarly, there are 3,124 INTxx2 descriptors per record template for the DB7-DB9 databases, since the number of clusters doubles again from the medium to large database set.

With these factors taken into consideration, the system evaluator may apply this methodology using the steps presented above to develop the descriptor ranges for each test configuration and associated database. Now, let us turn our attention to the test-transaction mix to be used with this test-database set for measuring the performance of the multi-backend database system MBDS.

# VI. THE TEST-TRANSACTION MIX

In this chapter we develop a test-transaction mix for benchmarking the multi-backend database system (MBDS). In the first section we review the test objectives to be satisfied. We present the test-transaction mix in the second section, and we discuss the test methodology, and present a test sequence which minimizes the loading and reloading of the test-database files in the third section. Finally, we end the chapter with a discussion of other test considerations which may simplify the test process of prospective system evaluators.

## A. THE TEST OBJECTIVES

The test-database files that we designed in Chapter V, and the test-transaction mix that we present in this chapter are intended for future application in a comprehensive performance evaluation of MBDS. This benchmarking effort will attempt to verify the performance-gain and capacity-growth claims of MBDS which we have described and analyzed in Chapter II. A second, equally important objective is to measure the overall system performance of MBDS.

MBDS is designed especially to process very large databases. The test-database set we present in Chapter V provides database files with as few as 9,372 records for the 2,000-byte record class of DB1, up to the largest file of the set which exceeds one million records for the 200-byte record class of the DB9 database. These test-database sets should provide an ample data source for the MBDS benchmark analysis.

We anticipate that the primary operation to be performed on a system such as MBDS will be to retrieve data from the applicable data store. Therefore, tests which focus on the RETRIEVE request will provide useful data for verifying the performance-gain and capacity-growth claims. To measure the overall MBDS performance, we propose a test-transaction mix which includes a complete set of the five MBDS operations, i.e., DELETE, INSERT, RETRIEVE, RETRIEVE-COMMON, and UPDATE requests.

The RETRIEVE and DELETE requests have very similar processing steps. Let us first consider the RETRIEVE request. Following the descriptor search, cluster search, and address generation activities, the record processing process in the backends fetches the selected records from the secondary storage. Record processing selects from the staged data set the records that satisfy the query, extracts the relevant values from the selected records, performs the required aggregate operation(s), and then forwards the results to the post processing process in the controller [Ref. 16: pp. 34-36].

MBDS follows almost the same steps for the DELETE operation. Following the descriptor search, cluster search, and address generation activities, record processing fetches the selected records from the secondary storage. Record processing selects from the staged data set the records that satisfy the query, marks the selected records for deletion, and then writes them back out to the secondary storage. Record processing then sends a completion message to the post processing process in the controller [Ref. 16: pp. 32-34]. We expect that the RETRIEVE and DELETE requests will provide important statistics for verifying the performance-gain and capacity-growth claims. Therefore, we design a diverse mixture of RETRIEVE and DELETE requests, to include overhead-intensive and data-intensive queries, as discussed in the last section of Chapter III.

The RETRIEVE-COMMON requests provide the opportunity to test multi-file requests. We design RETRIEVE-COMMON requests for the 2000-byte and 1000-byte record files of the DB1 database. (See Table 61.) Logically, the record processing process will handle two RETRIEVE requests, and fetches two sets of selected records from the secondary storage. Record processing then selects from the two staged record sets in the primary memory the records which satisfy the query, and returns the results to the user via the controller [Ref. 3: pp. 15-16].

To test the MBDS INSERT request, we propose two sets of requests. One set inserts new records into **existing** clusters, while the second set inserts records into **new** clusters. Similarly, three types of UPDATE requests are possible with MBDS. One type of UPDATE request returns the modified records to the **same, existing** clusters. The second type of UPDATE causes the modified records to change clusters. The "old" records are deleted, and the "new" records are

inserted into **different**. **existing** clusters. or to **new** clusters. Finally. the third type of UPDATE request is a blend of the first two types. That is, some of the modified records stay in the same. existing clusters. while other records change clusters in the same manner as described above for the second type of UPDATE request. We include all three types of UPDATEs in our test-transaction mix.

## B. THE TEST-TRANSACTION MIX

Table 66 displays the query portion of our first three retrieval requests, while Table 67 represents an analysis of the workload incurred by the requests of Table 66. Let us briefly analyze the intent of each of these requests.

TABLE 66. REQUEST SET 1.

| Request Number: | RETRIEVAL Request Queries: |
|---|---|
| 1 | ((TEMPLATE = TEMP2000) and (INT2001 $\geqslant$ 121) and (INT2001 $\leqslant$ 132)) |
| 2 | (((TEMPLATE = TEMP2000) and (INT2001 $\geqslant$ 4,823) and (INT2001 $\leqslant$ 4,870)) or ((TEMPLATE = TEMP2000) and (INT2001 $\geqslant$ 6,087) and (INT2001 $\leqslant$ 6,122))) |
| 3 | ((TEMPLATE = TEMP2000) and (INT2002 $\leqslant$ 2,343)) |

TABLE 67. REQUEST SET 1 WORKLOAD.

| Request Number | Number of Clusters Examined | Volume of Database Accessed | Volume of Database Retrieved |
|---|---|---|---|
| 1 | 86 | 344 records | 12 records |
| 2 | 174 | 31.56% | 84 records |
| 3 | 339 | 25.09% | 25.00% |

Request 1 examines the small portion of the database represented by the attribute INT2001 and its descriptor-ID D2-1. (See Table 62.) This request stages 344 records from the secondary memory to the primary memory. However, only the 12 records from clusters C30, C31, and C32 are answers of the

request. Therefore, the request evaluates how well MBDS performs when it examines a small amount of data (344/9372 records, or 3.67% of the database), and retrieves only a small amount of data from the set examined (12/344 records, or 3.49%). We classify request 1 as overhead-intensive.

Request 2 is designed to examine a large portion of the database (31.56%), but to retrieve only a small portion of the data examined. Although the request stages 2,958 records from the secondary storage to the primary storage, only 84 records (48 from clusters C530, C531, and C532, and 36 from clusters C609 and C610) participate in the response set. Thus, this request evaluates how well MBDS performs when it retrieves only a small amount of data from a large amount of data (84/2958 records, or 2.84%) which must be examined. Although the amount of data retrieved is small, MBDS must access a large amount of data to satisfy the query. Therefore, we classify request 2 as data-intensive.

Request 3 retrieves 25% of the database. The request examines a large portion of the database (25.09%, or 2,352 records). Of the 2,352 records which are staged to the primary memory, 99.62% (2343/2352) are relevant to the response set. Therefore, this request evaluates how well MBDS performs when nearly all of the data examined is retrieved to satisfy the query. We classify request 3 as data-intensive.

Table 68 displays the queries for requests 4, 5, and 6. These are all UPDATE requests which will return the updated records to their same, existing clusters. Table 69 depicts an analysis of the workload associated with each of these requests. The intent of requests 4, 5, and 6 is to update 1/8, 1/4, and 1/2 of the database, respectively.

Request 4 updates one-eighth of the database. The request causes 1.178 records from 212 clusters to be staged from the secondary memory to the primary memory. Then, 1.172 records (1/8 of 9.372) have the values of the attribute STRING001 changed to the character-string value OneEighth. These records are then returned to their original, existing clusters in the secondary storage. This request evaluates how well MBDS performs when nearly all of the data accessed (1172/1178 records, or 99.49%) is updated. Since most of the workload for this

TABLE 68. REQUEST SET 2.

| Request Number: | UPDATE Request Queries: |
|---|---|
| 4 | ((TEMPLATE = TEMP2000) and (INT2002 ⩽ 1,172)) (STRING001 = OneEighth) |
| 5 | ((TEMPLATE = TEMP2000) and (INT2002 ⩽ 2,343)) (STRING005 = OneQuartr) |
| 6 | ((TEMPLATE = TEMP2000) and (INT2002 > 4,686)) (STRING010 = One-Half) |

TABLE 69. REQUEST SET 2 WORKLOAD.

| Request Number | Number of Clusters Examined | Volume of Database Accessed | Volume of Database Updated |
|---|---|---|---|
| 4 | 212 | 12.57% | 12.50% |
| 5 | 339 | 25.09% | 25.00% |
| 6 | 261 | 50.06% | 50.00% |

request involves accessing and processing data records, we classify request 4 as data-intensive.

Request 5 updates one-quarter of the database. With this request, 2,343 of the 2,352 records accessed are updated and returned to the same, existing clusters in the secondary memory. This request updates the values of the attribute STRING005 to the new character-string value One-Quartr. Similarly, request 6 updates one-half of the database. The request updates 4,686 of the 4,692 records accessed. and returns the records to their original, existing clusters in the secondary storage. (Request 6 changes the STRING010 value to One-Half.) We classify requests 5 and 6 as data-intensive.

Requests 7 through 11. depicted in Table 70. are all RETRIEVE requests which are designed to access the updated records generated by requests 4. 5. and 6. Table 71 shows the corresponding workload statistics for requests 7 through 11.

TABLE 70. REQUEST SET 3

| Request Number: | RETRIEVAL Request Queries: |
|---|---|
| 7 | ((TEMPLATE = TEMP2000) and (INT2001 $\leq$ 4,686) and (STRING001 = OneEighth)) |
| 8 | ((TEMPLATE = TEMP2000) and (STRING001 = OneEighth)) |
| 9 | ((TEMPLATE = TEMP2000) and (STRING005 = OneQuartr)) |
| 10 | ((TEMPLATE = TEMP2000) and (STRING010 = One-Half)) |
| 11 | ((TEMPLATE = TEMP2000) and (INT2002 $\geq$ 4,687) and (STRING010 = One-Half)) |

TABLE 71. REQUEST SET 3 WORKLOAD.

| Request Number | Number of Clusters Examined | Volume of Database Accessed | Volume of Database Retrieved |
|---|---|---|---|
| 7 | 521 | 50.09% | 12.51% |
| 8 | 781 | 100.00% | 12.51% |
| 9 | 781 | 100.00% | 25.00% |
| 10 | 781 | 100.00% | 50.00% |
| 11 | 261 | 50.06% | 50.00% |

Requests 7, 8, and 9 are used to gauge MBDS performance when only a portion of the staged data is relevant to the response set. Request 7 accesses 50.09% of the database (4,694 records), of which 24.97%, or 1,172 records which have the attribute-value pair <STRING001, OneEighth> are included in the response set. Request 8 accesses 100% of the database, of which 12.51%, or 1,172 records are relevant to the answer. Request 9 accesses 100% of the database, of which 25%, or 2,343 records have <STRING005, One-Quartr> in the records of the response set. We classify all three of these requests as data-intensive.

Request 10 is designed to measure how well MBDS performs when 50% of the accessed data is relevant. While all 9,372 records in the database are staged to the primary memory, only the 4,686 records whose (attribute) STRING010

values are One-Half are included in the response set. Finally, request 11 gauges MBDS performance when almost all of the data staged to the primary memory participates in the response set. Of the 4,692 records accessed, 99.87% (4,686/4,692) are relevant to the answer. We classify requests 10 and 11 as data-intensive.

Table 72 depicts the request specifications for requests 12, 13, and 14, which are all RETRIEVE-COMMON requests. The corresponding workload statistics are shown in Table 73. We interpret Request 12 as follows. The first RETRIEVE request on the 2000-byte record file of database DB1 is called the source request. This source request causes 344 records to be staged from the secondary memory to the primary memory. The 12 records which satisfy this source request are retrieved and stored in a buffer area which we refer to as the source record set.

The second RETRIEVE request, which retrieves records from the 1000-byte record file, is called the target request. When it processes this target request, MBDS stages 688 records to the primary memory. MBDS selects the 264 records which satisfy the target request query and saves them in a second buffer area which we call the target record set.

Finally, MBDS does a pairwise merge operation between the records of the source and target record sets. During this merge, MBDS selects the 12 records from the source and target record sets which share common INT2001 and INT1001 attribute values, and returns them to the user via the controller [Ref. 19: pp. 27-32]. Note that we retrieve the smallest number of records from the source file, while the larger file to be searched against is designated as the target file. This feature is intrinsic to an efficient merge operation. The purpose of request 12 is to gauge MBDS performance when it examines a small amount of data for both the source and target requests, for which only a small amount of the staged data is relevant to the answer. Relative to the next two RETRIEVE-COMMON requests, request 12 may be categorized as an overhead-intensive request.

TABLE 72. REQUEST SET 4.

| Request Number: | RETRIEVE-COMMON Request Specifications: |
|---|---|
| 12 | RETRIEVE ((TEMPLATE = TEMP2000) and (INT2001 ≥ 121) and (INT2001 ≤ 132)) (INT2001)<br><br>COMMON(INT2001, INT1001)<br><br>RETRIEVE ((TEMPLATE = TEMP1000) and (INT1001 ≤ 264)) (INT1001) |
| 13 | RETRIEVE ((TEMPLATE = TEMP2000) and (STRING010 = One-Half)) (INT2002)<br><br>COMMON(INT2001, INT1001)<br><br>RETRIEVE ((TEMPLATE = TEMP1000) and (STRING010 = One-Half)) (INT1002) |
| 14 | RETRIEVE ((TEMPLATE = TEMP2000) and (INT2001 ≤ 4,686)) (INT2001)<br><br>COMMON(INT2002, INT1002)<br><br>RETRIEVE ((TEMPLATE = TEMP1000) and (INT1001 ≥ 3,515) and (INT1001 ≤ 4,686)) (INT1001) |

TABLE 73. REQUEST SET 4 WORKLOAD.

| Request Number | Number of Clusters Examined by the Source Request | Number of Records Accessed by the Source Request | Number of Records Relevant to the Source Request | Number of Clusters Examined by the Target Request | Number of Records Accessed by the Target Request | Number of Records Relevant to the Target Request | Size of the Result Record Set in Records |
|---|---|---|---|---|---|---|---|
| 12 | 86 | 344 | 12 | 86 | 688 | 264 | 12 |
| 13 | 781 | 9,372 | 4,686 | 781 | 18,744 | 9,372 | 4,686 |
| 14 | 261 | 4,692 | 4,686 | 87 | 1,740 | 1,172 | 1,172 |

The source request for request 13 causes all 9,372 records to be accessed from the 2000-byte record file. Of these, 4,686 records (50%) are relevant to the source request query, and are selected for insertion into the source record set. The target request accesses all 18,744 records from the 1000-byte record file, of which 9,372 records (50%) are relevant to the target request query, and are selected for the target record set. MBDS performs the merge operation between the source and target record sets, and returns the 4,686 records which have common INT2001 and INT1001 attribute values to the user via the controller. The purpose of this request is to see how well MBDS performs a RETRIEVE-COMMON operation which stages large quantities of data to the secondary memory, for which 50% of the staged data is relevant for both the source request and the target request. Thus, request 13 exemplifies a data-intensive query, which also experiences a significant amount of overhead in processing the request.

The number of records in the source record set for requests 12 and 13 directly correspond to the relevant data to return to the user. We assume the opposite approach with request 14. The source request for request 14 causes 4,692 records from the 2000-byte record file to be staged to the primary memory. Of these records, 4,686 are relevant to the source request query, and enter into the source record set. The target request stages 1,740 records from the 1000-byte record file, of which 1,172 records are relevant to the target query. (In effect, we force MBDS to execute an inefficient merge operation by using a source record set which is much larger than the target record set.) As a result of the merge operation on the source and target record sets, the 1,172 records which share common INT2002 and INT1002 attribute values are returned to the user via the controller. Request 14 gauges MBDS performance for the case where nearly all of the records staged for the source request are relevant to the source request, while only 25% of the records staged for the target request are relevant. We categorize request 14 as being an overhead-intensive, data-intensive request.

Table 74 shows the request specifications for requests 15 and 16, which are both INSERT requests. Recall from Chapter IV that the MBDS controller directs the insertion of new records by designating a specific backend to insert the new record into its secondary storage. The intent of requests 15 and 16 is to see

if a single INSERT request experiences a response-time variance as the number of backends in the test configuration increases. Request 15 inserts a record into an existing cluster (C1), while request 16 inserts a record into a **new** cluster.

TABLE 74. REQUEST SET 5.

| Request Number: | INSERT Request Specifications: |
|---|---|
| 15 | (<TEMPLATE,TEMP2000>,<INT2001,1>,(INT2002,1>,<MULTIPLE,Four>, <STRING001,Xxxxxxxxx>, ..., <STRING196,Xxxxxxxxx>) |
| 16 | (<TEMPLATE,TEMP2000>,<INT2001,1>,(INT2002,400>,<MULTIPLE,One>, <STRING001,Xxxxxxxxx>, ..., <STRING196,Xxxxxxxxx>) |

TABLE 75. REQUEST SET 5 WORKLOAD.

| Request Number | Number of Clusters Examined | Volume of Database Accessed | Volume of Database Inserted |
|---|---|---|---|
| 15 | - | - | 1 record |
| 16 | - | - | 1 record |

We expect to be able to note performance-gain statistics from DELETE requests which will be comparable to those experienced by RETRIEVE requests, since the processing steps associated with each of these database operations are very similar. Consequently, we select the eight DELETE requests shown in Table 76 which are designed to imitate the workload performed by the RETRIEVE requests 1 through 3, and 7 through 11 above. Table 77 depicts the workload analysis corresponding to these DELETE operations.

The DELETE operation for request 17 maps back to the workload of request 1. Request 17 will cause MBDS to stage 344 records to the primary memory, but will only delete the 12 records from clusters C30, C32, and C32. Therefore, this request gauges MBDS performance when it examines a small amount of data (344/9,372 records), and deletes only a small amount of data from the set

TABLE 76. REQUEST SET 6

| Request Number: | DELETE Request. Queries: |
|---|---|
| 17 | ((TEMPLATE = TEMP2000) and (INT2001 $\geqslant$ 121) and (INT2001 $\leqslant$ 132)) |
| 18 | ( ( (TEMPLATE = TEMP2000) and (INT2001 $\geqslant$ 4.823)and (INT2001 $\leqslant$ 4.870)) or ((TEMPLATE = TEMP2000) and (INT2001 $\geqslant$ 6,087)and (INT2001 $\leqslant$ 6,122))) |
| 19 | ((TEMPLATE = TEMP2000) and (INT2002 $\geqslant$ 7,030)) |
| 20 | ((TEMPLATE = TEMP2000) and (INT2001 $\leqslant$ 4,686) and (STRING001 = OneEighth)) |
| 21 | ((TEMPLATE = TEMP2000) and (STRING001 = OneEighth)) |
| 22 | ((TEMPLATE = TEMP2000) and (STRING005 = OneQuartr)) |
| 23 | ((TEMPLATE = TEMP2000) and (STRING010 = One-Half)) |
| 24 | ((TEMPLATE = TEMP2000) and (INT2002 $\geqslant$ 4,687) and (STRING010 = One-Half)) |

TABLE 77. REQUEST SET 6 WORKLOAD.

| Request Number | Number of Clusters Examined | Volume of Database Accessed | Volume of Database Deleted |
|---|---|---|---|
| 17 | 86 | 344 records | 12 records |
| 18 | 174 | 31.56% | 84 records |
| 19 | 121 | 25.07% | 25.00% |
| 20 | 521 | 50.09% | 12.51% |
| 21 | 781 | 100.00% | 12.51% |
| 22 | 781 | 100.00% | 25.00% |
| 23 | 781 | 100.00% | 50.00% |
| 24 | 261 | 50.06% | 50.00% |

examined (12/344 records). We classify request 17 as primarily overhead-intensive.

Similarly, request 18 corresponds to the workload of request 2. Request 18 stages 2,958 records to the primary memory, but only deletes 84 of the records accessed. Thus, the request evaluates how well MBDS performs when it deletes only a small amount of data from a large amount of data which must be accessed (84/2958 records, or 2.84%). We classify request 18 as both overhead-intensive and data-intensive, since it must examine a large-number of records, although only a small number of records are relevant to the answer.

Request 19 is a DELETE operation which corresponds to the request 3 workload. Request 19 causes MBDS to examine a large portion of the database (25.09%, or 2,352 records), and delete 99.62% (2,343/2,352) of the records examined. Thus, request 19 gauges MBDS performance when nearly all of the data examined is deleted. Request 19 is a data-intensive request.

Requests 20, 21, and 22 are the DELETE operations which are equivalent to requests 7, 8, and 9, respectively. Each of these DELETEs are used to measure MBDS performance when only a portion of the staged data is to be deleted. Request 20 causes MBDS to access 50.09% of the database, and delete 1,172 records, or 24.97% of the data accessed (1,172/4,964 records). Request 21 accesses 100% of the database, and deletes 1,172 records, or 12.51% of the data examined (1,172/9,372). Finally, request 22 accesses 100% of the database, and deletes 2,343 records, or 25% of the data examined (2,343/9,372 records). We classify requests 20, 21, and 22 as data-intensive requests.

Requests 23 and 24 are the DELETE operation equivalents of the RETRIEVE operations performed by requests 10 and 11. Request 23 deletes 50% (4,686/9,372) of the data which is staged to the primary memory, while request 24 deletes 99.87% of the data accessed (4,686/4,692 records). We classify both of these requests as data-intensive.

Table 78 specifies the queries for our next set of UPDATE requests, while Table 79 depicts the corresponding workload analysis. Request 25 will cause MBDS to update 12 records, causing the records to switch to brand new clusters. Therefore, the 12 "old" records will be deleted from the existing clusters, and the

## TABLE 78. REQUEST SET 7.

| Request Number: | UPDATE Request Queries: |
|---|---|
| 25 | ((TEMPLATE = TEMP2000) and (INT2001 $\geqslant$ 121) and (INT2001 $\leqslant$ 132))<br>(INT2001 = INT2001 + 2,312) |
| 26 | ((TEMPLATE = TEMP2000) and (INT2002 $\leqslant$ 2,343))<br>(INT2001 = INT2001 + 4,694) |
| 27 | ((TEMPLATE = TEMP2000) and (INT2002 > 7,653) and (INT2002 $\leqslant$ 9,352))<br>(INT2002 = INT2002 + 20) |
| 28 | ((TEMPLATE = TEMP2000) and (INT2002 > 3,477) and (INT2002 $\leqslant$ 3,504))<br>(INT2002 = INT2002 + 14) |
| 29 | ((TEMPLATE = TEMP2000) and (INT2002 > 5,287) and (INT2002 $\leqslant$ 5,350))<br>(INT2002 = INT2002 + 8) |
| 30 | ((TEMPLATE = TEMP2000) and (INT2001 >. 7,029))<br>(INT2002 = INT2002 + 10) |

## TABLE 79. REQUEST SET 7 WORKLOAD.

| Request Number | Number of Clusters Examined | Volume of Database Accessed | Volume of Database Updated |
|---|---|---|---|
| 25 | 86 | 344 records | 12 records |
| 26 | 339 | 25.09% | 25.00% |
| 27 | 86 | 18.35% | 18.14% |
| 28 | 2 | 28 records | 28 records |
| 29 | 4 | 64 records | 64 records |
| 30 | 172 | 35.06% | 25.00% |

12 "new" records will be inserted into newly created clusters. This request will gauge how well MBDS performs when it must examine a small amount of data (344/9372 records), and update a small amount of data from the set accessed (12/344 records), resulting in 12 record deletions and 12 record insertions. We classify request 25 as overhead-intensive.

Request 26 is designed to update 25% of the database, causing the records to migrate to brand new clusters. This request will cause 2,352 records to be staged into the primary memory. Of these, 2,343, or 99.62% (2,343/2,352) will be updated. This will result in 2,343 record deletions, accompanied by an identical number of record insertions into newly created clusters. Thus, the request will test MBDS performance when it must access a large amount of data, and then update nearly all of the accessed records, resulting in a sizable migration of records into newly created clusters. We classify request 26 as data-intensive.

In contrast, the UPDATE operations of requests 27 and 28 are designed to cause a migration of records into **existing** clusters. Request 27 accesses 1,720 records, and causes 1,700 records, or 98.84% of the records examined to switch to different, existing clusters. Therefore, MBDS will delete 1,700 "old" records, and insert 1,700 "new" records into existing clusters. Request 27 is a data-intensive request. Request 28 causes MBDS to examine just 28 records. However, all 28 records are updated, and forced to migrate to different, existing clusters. Request 28 is primarily overhead-intensive.

Our last two UPDATE operations are performed by requests 29 and 30. The purpose of these requests is to have some records remain in the same cluster, some migrate to different, existing clusters, and others migrate to newly created clusters. Request 29 causes MBDS to examine just 64 records. However, all 64 records accessed are updated. One-half of the updated records remain in their same, existing clusters, while the others migrate to different, existing clusters. Request 29 is primarily overhead-intensive.

Finally, request 30 updates 25% (2,343/9,372 records) of the database. This request stages 3,286 records to the primary memory. Of these staged records, 2,343, or 71.30% (2,343/3,286) are updated. Some of these records stay in the same cluster, others migrate to different, existing clusters, while the last 10

records migrate to a newly created cluster. We classify request 30 as data-intensive.

The system evaluator should note that the requests we include in this test-transaction mix are described only for the DB1 database of Table 61, which is used for test configurations 1, 2, and 3 for the small-size database set. However, the same transactions may be used to test with the DB2 and DB3 databases. Requests 15 and 16 will only insert 1 record each, regardless of the test database being used. However, the number of records affected by the other requests changes as we change to a different test database.

Although the number of records doubles from DB1 to DB2. and triples from DB1 to DB3. the INT2001 and INT 2002 attribute value ranges remain the same. The MULTIPLE attribute acts to produce two unique records for each pair of INT2001 and INT2002 attributes for the DB1 database, and three unique records for each pair of INT2001 and INT2002 attributes for the DB3 database. Since the requests of the test-transaction mix all key on the INT2001/INT2002 attribute values, the effect is that the number of records retrieved by request 1, for example, will double to 24 with the DB2 database, and triple to 36 with the DB3 database. Similar changes occur with the number of records retrieved, deleted, or updated by the other test transactions.

Therefore, we have achieved the effect of increasing the response set size in the same proportion to corresponding increases in the database size, using the same set of requests from the test-transaction mix. Also, as claimed in the last section of Chapter III, we have a test-record organization, a test-database structure, and a test-transaction mix set which enables the system evaluator to use the same organization. structure. and mix for all system configurations for a particular database size category **without modification!**

The system evaluator must also keep the following factors in mind. The test-transactions presented in this chapter must be run for all four record files for each test-database set. for all three database sizes (small. medium. and large), and for all five configurations (when testing a system with a maximum of four backends). Since the same set of requests may be used for all system

configurations for a given database size category, we require 12 different sets of requests (one each for each record file, per database size category).

Obviously, the required number of test iterations grows considerably if a system with more than 4 backends is to be tested. Therefore, the system evaluator must choose to balance the amount of work required to conduct a complete benchmark test against the benefits to be derived. A carefully chosen subset of this test-transaction mix can provide a quick estimation of the performance-gain and capacity-growth potential provided by the MBDS.

## C. THE TEST SEQUENCE

The ordering and sequencing of the benchmarks is an important factor to consider. In Figure 19 we present one scheme to sequence the requests and minimize the need to reload the database.

Requests 1 through 20 may be executed in sequence. Executing request 20 after request 17 will mean that request 20 will delete 1,160 records instead of 1,172 records, since 12 records in the relevant record set domain are deleted by request 17. This does not influence the test, since the intent of request 20 remains intact.

Requests 21 through 30 do affect each other, since the various DELETE and UPDATE operations act on overlapping record sets. Therefore, we propose executing the requests separately, as shown in Figure 19.

The system evaluator may decide to reduce the size of the test-transaction mix to reduce the amount of work required. A judiciously chosen subset of the test-transaction mix may be used to conduct system testing. It may then become feasible to resequence the subset of requests to hopefully reduce the number of times to load and reload the database.

## D. OTHER TEST CONSIDERATIONS

In Chapter I we discussed the performance-measurement tools developed by Kovalchik [Ref. 5], and the external and internal timing checkpoints which have been embedded in the MBDS code by Tekampe and Watson [Ref. 6]. To conduct system testing with the test-transaction mix and test-database set we propose in this thesis, we recommend one modification to an external database

| | |
|---|---|
| Step 1. | Load the four database DB1 record files. |
| Step 2. | Execute requests 1 through 20 in order. |
| Step 3. | Repeat step 1 above. |
| Step 4. | Execute requests 4, 5, 6, 25, 26, 27, 28, 29, 21, and 23 in the order listed. |
| Step 5. | Repeat step 1 above. |
| Step 6. | Execute request 4, 5, 6, 30, 22, and 24 in the order listed. |

Figure 19. Proposed Test Execution Sequence.

creation program written by Tekampe and Watson, namely **performance database load** (perdbld.c). In its present mode, this program can only create record files with a maximum of 1000-records, for a fixed record format of 33 6-byte attributes. The program is not interactive, and can not create more than one file per run.

To be useful for future MBDS benchmarking efforts, the following enhancements are proposed for this program. First, make the program interactive. This will enable the user to specify the four file names and the number of records per file interactively, eliminating the need to recompile the program each time a new test-database is required. The four record templates specified in Chapter V (see Table 58 again) can be formated in the code. Finally, upper limits in sizes of the largest files per record class for DB9 can be used as size parameters within the code.

To create a specific test database (DB1 through DB9), the system evaluator would run the "perdbld" program, and enter the corresponding file names and number of records for each file. The program would create four files, (one each for the 2000, 1000, 400, and 200-byte record classes), in a format which can be used as input for the test-interface (TI) controller process. This would greatly

streamline the test-database creation process. and would enable the system evaluator to generate test files as needed, rather than tying up disk space with preformated test-files for all nine test-database sets.

To conduct system testing, the evaluator uses the test-interface (TI) controller process to load test files and create required directory entries. This process also provides a means to generate test-transaction requests which may be executed and/or archived for later use. In our experience, this feature appears to be far to slow to be useful for large-scale system testing. A much simpler scheme is to create text files containing the desired test transaction(s). These files are formated in the exact manner as those created by the test-interface process for archived requests. Instead of having it input requests which TI has saved in archived files, TI will read the text files containing the desired test-transactions. This scheme is much simpler, and saves the system evaluator from having to respond to a complex sequence of interactive menus to create the desired request files.

From our experience with the prototype MBDS running on the VAX/PDP-11/44 environment, we have compiled an abridged testplan checklist to assist system testers and users with system operation. This checklist is included as Appendix A. The actual steps involved in testing will change with the conversion to the new Sun/Unix configuration. However, this checklist should prove useful by serving as a guide for the format of a detailed testplan for future test efforts.

As noted in the checklist of Appendix A, the TI-process menus provide the system evaluator with a flexible set of processing flags which may be set on/off as desired to enable processing without timing measurements, with external measurements only, or with both external and internal measurements. Recall from Chapter I that the external measurement facility provides a measure of the response time of a request, while the internal measurement facility permits evaluation at the microscopic level. By observing the internal performance of the system software, we can analyze the system's work distribution. Our goal here is to be able to identify code segments which may be candidates for fine-tuning to further enhance system performance.

We anticipate that initial testing will be done with external measurements only. This will enable the system evaluators to gain experience with system operation. It should also provide sufficient detailed test data to enable the system evaluators to verify the performance-gain and capacity-growth claims. Analysis of the data provided by the tests conducted with external measurements only should provide indications as to where the system evaluators should concentrate their efforts regarding testing at the microscopic level with internal measurements. For example, some transactions will spend a lot of time in the backend record processing process. The system evaluators may repeat an appropriate subset of the test-transaction mix with the various record-processing-timing-flags set.

Benchmarking is an experimental, "modify-on-the-fly" activity. While it is important for the benchmark tests to be machine, application, and database independent, it may be necessary to refine and redefine some of the benchmarks during the performance evaluation process. Therefore, the test-transaction mix presented in this chapter is not a "hard-and-fast" mix. Consequently, we plan to benchmark MBDS with the test-transactions and the test-database organizations not only on the first set of new MBDS hardware, i.e., ten Sun (Unix) workstations, but also on a second set of new MBDS hardware, i.e., a large number of MicroVAX-II (VMS) systems.

# VII. THE CONCLUSION

In this thesis, we have analyzed the performance-gain and capacity-growth claims of software multiple-backend database systems. Our analysis of the performance-gain claim in terms of the resulting response-time reduction, enabled us to pose two logical questions. First, at what number (n) of backends will the response-time reduction stop? Second, how large will n be when the system overhead becomes pronounced? One of the goals of our future work in benchmarking the multi-backend database system, MBDS, will be to determine answers to these questions via empirical performance measurements. Our analysis of the capacity-growth claim in terms of the resulting response-time invariance, led us to the conclusion that we must select a test-database set and a test-transaction mix which enables us to easily increase database size with corresponding increases in the response set size. Thus, we can ask the question whether or not the response time of the system remains invariant when the number of backends is increased proportionally to the size of the response sets.

The analysis of the performance-gain and capacity-growth claims also enabled us to identify key design features for specifying a test-database set. From our analysis of the performance-gain claim, we conclude that we must develop a database sizing methodology which permits us to split the database into equal subsets to distribute among all of the backends, for all possible system configurations. This design factor led to our development of the database size multiple relation of Table 2. From our analysis of the capacity-growth claim, we design the MBDS test-database set in Chapter V to include the MULTIPLE attribute of Table 58, and the test-transaction mix design of Chapter VI.

Finally, our Chapter II analysis has led us to develop the Chapter III relationship of (2M - 1), which enables us to quickly determine the total number of test configurations required to test a system with M backends. With these basic design features, we develop a general methodology for designing a test-database set, including selection of record sizes, which is machine-independent

and application-independent. and which satisfies all of the required test system configurations.

We have applied the methodology to develop a test-database set and a test-transaction mix for the multi-backend database system, MBDS. Using record sizes of 2000, 1000, 400. and 200-bytes per record for demonstration purposes, we developed a sample test-database set for an MBDS with a maximum of four backends. By adhering to the methodology presented in Chapters III and V, the system evaluator may develop a test-database set for any system configuration, using any brand of hardware. Consequently, we have achieved a machine-independent design. The "synthetic" database format we have presented is a general database set which is independent of any real application. Furthermore, the test-transaction mix we present in Chapter VI to test system performance with the test database model is free from any specific real-world application. Therefore, we have attained database-independence and application-independence.

The test-transaction mix we have presented effects a comprehensive test of the five MBDS ABDL database operations. We believe that these test-transactions provide a complete set of requests to verify the system's performance-gain and capacity-growth claims, and to gauge overall system performance. Indeed, future system evaluators may find it most beneficial to select a judicious subset of the requests presented in Chapter VI, especially for tests involving several backends. For example, benchmarking a system with a maximum of eight backends requires 15 configurations ((2 x 8) - 1). If we assume four record classes per database and three database sizes (small, medium, and large), then the test-transactions will be executed 180 times (15 x 4 x 3). Therefore, a carefully chosen subset of the test-transaction mix will enable system evaluators to minimize the actual amount of work involved in performing the benchmark, while still obtaining ample statistics for gauging system performance.

The next step is to apply our methodology for an actual benchmark analysis of the MBDS. This effort will begin as soon as the hardware installation and software conversion to the new Sun/Unix environment is completed. More distant plans project acquisition of yet another set of hardware, based on the

DEC MicroVAX-II, which will operate under the MicroVMS operating system. The MBDS benchmark evaluation will be repeated with this new set of hardware, providing us with MBDS performance statistics for two sets of hardware (Sun and MicroVAX), and two different operating systems (Unix and MicroVMS). These two performance evaluations should adequately verify the system's performance-gain and capacity-growth claims, and attest to the applicability of our machine-independent, database-independent, and application-independent methodology for database system performance measurements.

Future MBDS benchmarking should also include an analysis of the impact of the breadth and depth of the MBDS directory structure on system performance. This research should measure the effects that varying the number of directory attributes, descriptor ranges, and cluster compositions have on system performance for a given workload. We believe that the basic test-database design methodology presented in this thesis may be easily extended to accommodate this research effort.

Finally, future benchmark efforts will be required to evaluate the four language interfaces being implemented as part of the research effort on multi-lingual database systems [Ref. 4]. This research extends MBDS by providing "transparent" user interfaces to the MBDS ABDL via the SQL, DL/I, Daplex, and CODASYL data manipulation languages. The results of these combined research efforts may well lead to entirely new vistas in the realm of database system research.

## APPENDIX A: TESTPLAN FOR MBDS TESTING.

**1. Backend: Setup and Initialization:**

1.1. Logon to PDP-11/44:
(via CRT to backend #1 - i.e., system 'A')

1.1.1. Enter appropriate user-id/password

1.2. Enter: "sh users" <return>
[to check to see if anyone else is logged on to the backend.]

1.2.1. System responds: "TT1: [6,16]"

1.2.2. Now, take the "write project" off of disk 0 (zero).

1.3. Enter: "run $shutup" <return>

1.3.1. System responds: "Enter minutes to wait before shutdown."

- enter: "0" <return> -- (i.e., a zero )

1.3.2. System responds: "Ok to shutdown? [y/n]"

- enter: "Y" <return> -- (i.e., yes)

<When the backend responds: "SHUTUP operation complete"
the PDP-11/44 will be shut-down.>

1.4. Now, change the plastic keys on the disk drives:

1.4.1. Make the left-hand drive 0, (zero).

1.4.2. Make the right-hand drive 1.

1.4.3. Write protect the right-hand drive,
(which is now logical-drive 1).

(Note: We will boot off of drive 0 - which is now the
left-hand drive, and contains the executable code.
Drive 1, which is now the right-hand drive, has source code).

1.5. On TTY, enter: "b db" <return>

1.5.1. TTY will ask for the date:
- Enter: <return>

1.5.2. When done booting-up, the backend system returns an EOF.
- Enter: "bye" <return>

1.5.3. TTY will log-off.

1.6. Logon to PDP-11/44 again - (via CRT)

1.6.1. Enter: "hel mdbs" <return>

1.6.2. Enter: "done" <return>

1.6.3. To list the files, enter: "pip/li" <return>

(note: .TSK - are exec files (abs))

1.6.4. To START the system, enter: "@run" <return>

1.6.5. To see what processes are running, enter: "par" <return>

1.6.6. To get to a different directory,
- enter: "set /uic = [ , ] " <return>
where:
[6,16] - are the external test flags
(has 1 record-processing-buffer = TB 0)

[6,17] - are the external & internal flags
(has 1 record-processing-buffer = TB 0)

[6,20] - are the external flags
(has 2 record-processing buffers = TB 0/TB 1)

**2. Controller: Setup and Initialization:**

2.1. Logon to the Vax 11/780: "DEMURJIAN" / (password)

2.2. Enter: "u" <return>

"u" => Eunice, which emulates UNIX, vi, etc,
on the VAX/VMS system.

2.3. Enter: "Jim" <return>

This command changes the directory to:
/work/demurjian/Watson/MDBS/RUNMDBS

- tests will be done on VerE.4 / TI = Test Interface

- See directory for files: RUNEXT, RUNINT
  - (which contain the task files: dblti.out*, gpcl.out*, iig.out*, pp.out*, ppcl.out*, reqprep.out*)

2.4. Now, decide whether you want to conduct external or internal tests:

2.4.1. To conduct external tests:

- Enter: "cp ../RUNEXT/* ." <return>
  (This copies task files dblti.out*, gpcl.out*, iig.out*, pp.out*, ppcl.out*, and reqprep.out* to the RUNMDBS directory).

2.4.2. To conduct internal tests:

- Enter: "cp ../RUNINT/* ." <return>
  (This copies task files dblti.out*, gpcl.out*, iig.out*, pp.out*, ppcl.out*, and reqprep.out* to the RUNMDBS directory).

2.5. To run, we must first quit Eunice.

2.5.1. Enter: "^D" <return> -- (i.e., "control D" <return>)
or Enter: "logout" <return>

2.5.2. Enter: "mdbs" <return>

- (This starts the MBDS controller processes on the VAX)

- MBDS is now "up" and ready for testing!

## 3. MBDS Test Procedures:

3.1. Enter: "run dblti.out" <return>
- (see code in TI/dblti.c)

3.2. MBDS responds: "How many backends are there? (1,2....)>"

- enter: "1/2" <return> as appropriate;

3.3. MBDS responds: "Do you want de-bugging messages printed? (y/n)>"

- enter: "y/n" <return> -- (use "n" for testing)

3.4. MBDS responds: "What operation would you like to perform? "
" (g) - generate database "
" (l) - load database "
" (e) - execute test interface "
" (x) - exit to operating system "
" (z) - exit and stop MDBS "
(for 1 BE only)

3.4.1. If "g" is selected in step 3.4., then /* generate database */

- A submenu follows to permit you to generate a db.

- DO NOT use for testing - takes TOO MUCH TIME!! INSTEAD, select "l" to load a db which we create beforehand.

3.4.2. If "l" is selected in step 3.4., then /* load database */

3.4.2.1. MBDS responds:
"ENTER NAME OF FILE CONTAINING TEMPLATE INFORMATION:"

- Enter: "fname" <return>

Example: "st.f" <return>

Note: (t => template
d => descriptor
r => record )

Therefore: st.f = template file
sd.f = descriptor file
sr.f = record file (1000 records)

3.4.2.2. MBDS responds:
"ENTER NAME OF FILE CONTAINING THE DESCRIPTORS:"

- Enter: "fname" <return>
Example: "sd.f" <return>

3.4.2.3. MBDS responds:
"ENTER NAME OF FILE CONTAINING RECORDS TO BE LOADED:"

- Enter: "fname" <return>

Example: "testr.f" <return>
or "sr.f" <return>

<MBDS inputs the database record; for every 100 records input, it prints a "*" on CRT screen>

3.4.3. If "e" is selected in step 3.4.,
then /* execute test interface */

3.4.3.1. MBDS responds:
"Do you ALWAYS want to wait for responses? y/n"

- Enter: "y" <return>

3.4.3.2. MBDS responds:
"Enter the type of subsession you want:
"(r) REDIRECT OUTPUT; select output for answers"
"(d) NEW DATABASE; choose a new database"
"(n) NEW LIST; create a new list of traffic units"
"(m) MODIFY; modify an existing list of traffic units"
"(s) SELECT; select traffic units from an existing list"
" (or give new traffic units) for execution"
"(o) OLD LIST; execute all the traffic units in an existing list."
"(p) PERFORMANCE TESTING"
"(x) EXIT; return to generate, load, execute, or exit menu"
"Selection>"

- NOTE: we will only use options: 'r', 'p', 's' and 'x'

- Enter: "r/p/s/x" <return>

3.4.3.2.1. First, select "r" in step 3.4.3.2;

MBDS responds:
"Enter the appropriate number for the output form."
"(1) Send output to CRT only."
"(2) Send output to File only."
"(3) Send output to both CRT and File."
"(4) Do not display output."

- enter: "4" <return> -- (use option "4" for testing)

- MBDS returns to menu of 3.4.3.2 above.

3.4.3.2.2. Second, select "p" in step 3.4.3.2;

3.4.3.2.2.1. MBDS responds:
"What would you like to do?
"(e) Turn on external timer."
"(i) Turn on internal timer."
"(a) ABORT.. Abandon all requested actions."
"(x) Exit to previous menu."
"Selection>"

- Enter: "e/i/a/x" <return>

- When "x" is selected, return to menu of 3.4.3.2 above.

3.4.3.2.2.1.1. If you select "e" in step 3.4.3.2.2.1, then:

- MBDS responds: "External Timer On."
[ sets 'TIMER_ON = 1 ' ]

- MBDS returns to menu of 3.4.3.2.2.1 above.

3.4.3.2.2.1.2. If you select "i" in step 3.4.3.2.2.1,
then < INIT_TIMERS >

MBDS responds:
"Do you wish to time message handling procedures in:"
"(a) IIG"
"(b) ReqPrep"
"(c) PP"
"(d) CC"
"(e) DM"
"(f) RecProc"
"(x) Exit to previous menu"

"Selection>"

- Enter: "a/b/c/d/e/f/x" <return>

- When "x" is selected, return to menu
  of 3.4.3.2.2.1.

3.4.3.2.2.1.2.1. If "a" is selected in step 3.4.3.2.2.1.2.
then: <TIM_IIG>

MBDS responds:
"Do you want to time: "
"(a) All routines in entire process" [TIIGAllM]
"(b) LoadType-C" [TLdTyCM]
"(c) ClusId" [TClIdM]
"(d) ReqForNewDescId" [TReqFNeDeIdM]
"(x) Exit to previous menu"
"Selection> "

- enter: "a/b/c/d/x" <return>

- When "x" is selected, return to menu'
  of 3.4.3.2.2.1.2.

3.4.3.2.2.1.2.2. If "b" is selected in step 3.4.3.2.2.1.2,
then /* TIM_ReqP */

MBDS responds:
"Do you want to time: "
"(a) All routines in entire process" [TIReqpAllM]
"(b) RP_$ReqsWithErr_PP (Rec Template)" [TReqNotOKM]
"(c) RP_S$ReqCnt_PP" [TReqOK1ReqM]
"(d) RP_S$AggOps_PP" [TReqOKAggM]
"(e) REQUEST_COMPOSE" [TReqCompM]
"(f) RP_BROADCAST_REQS_ALL_DM" [TReqBroadM]
"(g) RP_S$ReqsWithErr_PP (Parser Error)" [TReqSynErrM]
"(h) RecChangedClus" [TReqChClM]
"(i) NoMoreGenIns" [TReqNMGIM]
"(x) Exit to previous menu"
"Selection> "

- Enter: "a/b/c/d/e/f/g/h/i/x" <return>

- When "x" is selected, return to menu of 3.4.3.2.2.1.2.

3.4.3.2.2.1.2.3. If "c" is selected in step 3.4.3.2.2.1.2,
then /* TIM_PP */

MBDS responds:
"Do you want to time: "
"(a) All routines in entire process" [TPPAllM]
"(b) ReqsWithErr" [TReqWErrM]
"(c) NoOfReqsInTrans" [TNoORITM]
"(d) AggOps" [TAggOpsM]
"(e) BC_Res" [TBCResM]
"(f) BC_AO_Res" [TBCAOResM]
"(x) Exit to previous menu"
"Selection> "

- Enter: "a/b/c/d/e/f/x" <return>

- When "x" is selected, return to menu of 3.4.3.2.2.1.2.

3.4.3.2.2.1.2.4. If "d" is selected in step 3.4.3.2.2.1.2,
then /* TIM_CC */

MBDS responds:
"Do you want to time: "
"(a) All routines in entire process" [TCCAllM]
"(b) CidsForTrafUnit" [TCiFoTrUnM]
"(c) TypeC_+AttrsTrafUnit" [TTyCAtTUM]
"(d) DidSetsTrafUit" [TDiSeTrUnM]
"(e) AttrRelease" [TAtRelM]
"(f) InsAllAttrsRelease" [TInAlAtReM]
"(g) DidSetsRelease" [TDiSeReM]
"(h) UpdFinished" [TUpFinM]
"(i) C_Request_Completion" [TRecpCpM]
"(x) Exit to previous menu"
"Selection> "

- Enter: "a/b/c/d/e/f/g/h/i/x" <return>

- When "x" is selected. return to menu
of 3.4.3.2.2.1.2.

3.4.3.2.2.1.2.5. If "e" is selected in step 3.4.3.2.2.1.2.
then /* TIM_DM */

MBDS responds:
"Do you want to time: "
"(a) All routines in entire process" [TDM_AllM]

"(b) ParsedTrafUnit" [TDM_PTUM]
"(c) NoMoreGenIns" [TDM_NMGEM]
"(d) BeNo" [TDM_BNM]
"(e) NewDesc" [TDM_NDM]
"(f) DescIds" [TDM_DIM]
"(g) ATM_Create" [TDM_DCM]
"(h) ATM_insert" [TDM_DA_IM]
"(i) Desc_add" [TDM_DD_AM]
"(j) Catchall" [TDM_DCAM]
"(k) AttrLocked" [TDM_ALM]
"(l) DiDSetsLocked" [TDM_L_DSM]
"(m) CidsLocked" [TDM_C_LM]
"(n) OldNewValues" [TDM_ONVM]
"(o) UpdFinished" [TDM_UFM]
"(x) Exit to previous menu"
"Selection> "

- Enter: "a/b/ ... /n/o/x" <return>

- When "x" is selected, return to menu of 3.4.3.2.2.1.2.

3.4.3.2.2.1.2.6. If "f" is selected in step 3.4.3.2.2.1.2, then /* TIM_RecP */

MBDS responds:
"Do you want to time: "
"(a) The entire process" [TRecpAllM]
"(b) All routines in entire process"
"(c) ReqDiskAddrs" [TReqDisAddrM]
"(d) ChangedClusRes" [TChClResM]
"(e) NoMoreGenIns" [TNoMoGeInM]
"(f) Fetch" [TFetchM]
"(g) OLD_REQ" [TOldReqM]
"(h) PIO_WRITE" [TPioWriteM]
"(i) PIO_READ" [TPioReadM]
"(j) Disk_IO" [TDiskIOM]
"(x) Exit to previous menu"
"Selection> "

- Enter: "a/b/ ... /i/j/x" <return>

- When "x" is selected, return to menu of 3.4.3.2.2.1.2.

3.4.3.2.2.1.2.7. If "x" is selected in step 3.4.3.2.2.1.2. then:

- MBDS returns you to the menu of 3.4.3.2.2.1. above.

3.4.3.2.2.1.3. If you select "a" in step 3.4.3.2.2.1,
then /* TI_INT_TEST */

- MBDS sets: Timer_msg_ptr = 0;
Timer_on = 0;

- MBDS returns you to the menu of 3.4.3.2.2.1.

3.4.3.2.2.1.4. If you select "x" in step 3.4.3.2.2.1. then:

- MBDS returns you to the menu of 3.4.3.2. above.

3.4.3.2.3. Third, select "s" in step 3.4.3.2. < TI_SELECT >

3.4.3.2.3.1. MBDS responds:
"Enter the name for the traffic unit file."
"It may be up to 13 characters long,including the .ext."
"Filenames may include only one ';' character"
"as the first character before the version number"
"File name> "

- Enter: "fname" <return>

Example: "pevalrets.f" <return>

3.4.3.2.3.2. Then, MBDS reads TU(s) from the file, and
responds: "List of executable traffic units"
" ... "

3.4.3.2.3.3. Next, MBDS responds:
"Select Options "
"(d) display the traffic units in the list"
"(n) enter a new traffic unit to be executed"
"(num) execute the traffic unit at [num] "
"(x) exit from this SELECT subsession"
"Option> "

- Enter: "d/n/num/x" <return>

3.4.3.2.3.3.1. If you select "d" in step 3.4.3.2.3.3., then:

- MBDS displays traffic-units,

- MBDS returns you to the menu of 3.4.3.2.3.3.

3.4.3.2.3.3.2. If you select "n" in step 3.4.3.2.3.3., then:

- TI_SELECT calls TI_traffic_unit_get
  to let you enter a new TU.
  (refer to code in tisubs.c)

- Then, MBDS returns to menu of 3.4.3.2.3.3.

3.4.3.2.3.3.3. If you input a number (num) in step 3.4.3.2.3.3., then:

- MBDS, opens file "timer.res"
- processes the TU
- closes file "timer.res"

- MBDS responds:
  "The starting time for this request was ..."
  "The stopping time for this request was ..."
  "The total elapsed time was ..."
  "The number of buffers used was ..."

- Then. MBDS returns you to the menu of
  step 3.4.3.2.3.3.

3.4.3.2.3.3.4. If you select "x" in step 3.4.3.2.3.3., then:

- MBDS returns to subsession menu of 3.4.3.2. above.

3.4.3.2.4. Finally, select "x" from menu of 3.4.3.2.

- MBDS returns you to the main menu of 3.4.

3.4.4. If "x" is selected in step 3.4, then /* exit to UNIX */

- Exit MBDS program and return control to operating system.
  (processes are still active! ... follow QUIT PROCEDURES
  in section 4 below to terminate test session completely.)

3.4.5. If "z" is selected in step 3.4, then
/* exit & stop MBDS */

- Select this option if you are only testing with one backend, and are exiting the system "gracefully."

- MBDS will quit and terminate all processes.

  (If using more than 1 backend, or if you terminated MBDS execution abnormally, select option 'x' instead).

## 4. QUIT PROCEDURES (Cleanup and termination):

4.1. If you are running with only 1 backend and are stopping gracefully. simply select operation 'z' under section 3.4. above.

4.2. If you are running with 2 or more backends, or if you are having software problems and need to abort the system manually, do the following:

4.2.1. [on the VAX 11/780]:

4.2.1.1. Enter: "@stop" <return>

- All MBDS processes on the VAX will terminate:

- wait for 10-15 minutes before continuing at the next step.

4.2.2. [on the backend (PDP-11/44)]:

4.2.2.1. Enter: "abo cc...." <return> -- (on the backend's CRT)

4.2.2.2. Enter: "@stop" <return>

4.2.2.3. Enter: "run $shutup" <return>

4.2.2.3.1. System responds: "Enter minutes to wait before shutdown."

4.2.2.3.2. Enter: "0" <return> -- (i.e., a zero )

4.2.2.4. System responds: "OK to shutdown? [y/n]"

- Enter: "y" <return> -- (i.e., yes)

< The backend system (pdp-11/44) is now shut-down. >

4.2.2.5. When the line-printer prints the prompt, switch the disk drives:

- Change the plastic keys on the disk drives:

  - make the left-hand drive #1.

  - make the right-hand drive #0.

4.2.2.6. Now, log back onto the backend via the teletype (TTY):

4.2.2.6.1. On TTY, enter: "b db" <return>

4.2.2.6.2. TTY will ask for the date;

4.2.2.6.3. Enter: " dd-mmm-yy hh:mm" <return>

- (example: "14-JAN-85 16:05" <return> )

4.2.2.7. When done booting-up, system returns an EOF on TTY.

4.2.2.7.1. Enter: "bye" <return>

4.2.2.8. TTY will log-off.

4.2.2.9. Now, shut-off the backend CRT for backend-number 1.

4.2.2.10. Finally, write protect the right-hand drive.
(which is now, logical-drive 0)

4.2.2.11. That's it! Have a nice day!!

## LIST OF REFERENCES


1. Schoderbek, P. P., *Management Systems*, John Wiley & Sons. Inc., 1967.
2. Date, C. J., *An Introduction to Database Systems*, Addison Wesley. 1982.
3. Naval Postgraduate School, Monterey, California, Technical Report. NPS52-85-002, *A Multi-Backend Database System for Performance Gains, Capacity Growth and Hardware Upgrade*, by S. A. Demurjian. et al., February 1985.
4. Naval Postgraduate School, Monterey. California. Technical Report. NPS52-85-001. *New Directions in Database-Systems Research and Development*, by S. A. Demurjian and D. K. Hsiao. February 1985.
5. Kovalchik. J. G., *Performance Evaluation Tools for a Multi-Backend Database System*, M.S. Thesis. Naval Postgraduate School, Monterey. California. December 1983.
6. Tekampe, R. C., and Watson, R. J., *Internal and External Performance Measurement Methodologies for Database Systems*, M.S. Thesis, Naval Postgraduate School. Monterey. California, June 1984.
7. The Ohio State University, Columbus, Ohio, Technical Report. OSU-CISRC-TR-81-7, *Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part I)*, by D. K. Hsiao and M. J. Menon, July 1981.
8. Naval Postgraduate School, Monterey, California. Technical Report. NPS52-84-023, *Performance Measurement Methodologies for Database Systems*, by S. A. Demurjian, et al., December 1984.
9. Naval Postgraduate School, Monterey, California. Technical Report. NPS52-84-005, *The Implementation of a Multi-Backend Database System (MBDS): Part IV - The Revised Concurrency Control and Directory Management Processes and the Revised Definitions of Inter-Process and Inter-Computer Messages*, by S. A. Demurjian, et al., March 1984.
10. Demurjian, S. A., et al., "Performance Evaluation of a Database System in Multiple Backend Configurations," *Proceedings of the 1985 International Workshop on Database Machines*, March 1985.
11. Demurjian, S. A. and Hsiao, D. K., "Benchmarking Database Systems in Multiple Backend Configurations," *IEEE Database Engineering Bulletin*, March 1985.
12. Strawser, P. R., *A Methodology for Benchmarking Relational Database Machines*, Ph.D. Dissertation, The Ohio State University, Columbus, Ohio. March 1984.
13. Hawthorn, P. B., and Stonebraker, M., "Performance Analysis of a Relational Data Base Management System," *Proceedings of the ACM SIGMOD Conference on Management of Data*, Boston, May 30-June 1, 1979.

14. The Ohio State University, Columbus, Ohio. Technical Report. OSU-CISRC-TR-81-8, *Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part II),* by D. K. Hsiao and M. J. Menon, July 1981.

15. The Ohio State University, Columbus, Ohio. Technical Report. OSU-CISRC-TR-82-1, *The Implementation of a Multi-Backend Database System (MBDS): Part I - Software Engineering Strategies and Efforts Towards A Prototype MBDS,* by D. S. Kerr, et al., January 1982; also appeared in *Advanced Database Machine Architecture,* Hsiao (ed.), Prentice Hall, 1983.

16. Naval Postgraduate School, Monterey, California, Technical Report. NPS-52-82-008. *The Implementation of a Multi-Backend Database System (MBDS): Part II - The First Prototype MBDS and the Software Engineering Experience,* X. He, et al., July 1982: also appeared in *Advanced Database Machine Architecture,* Hsiao (ed.), Prentice Hall. 1983.

17. Naval Postgraduate School, Monterey, California, Technical Report, NPS-52-83-003, *The Implementation of a Multi-Backend Database System (MBDS): Part III - The Message-Oriented Version with Concurrency Control and Secondary-Memory-Based Directory Management,* by R. Boyne, et al., March 1983.

18. Hsiao, D. K., and Harary, F., "A Formal System for Information Retrieval from Files," *Communications of the ACM,* Vol. 13, No. 2. February 1970; Corrigenda, Vol 13., No. 4, April 1970.

19. Tung, H., *Design, Analysis and Implementation of the Primary Operation, Retrieve-Common, of the Multi-Backend Database System (MBDS),* M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1985.

20. Kernigan. B., and Ritchie, D. M., *The C Programming Language,* Prentice-Hall, 1978.

215848

Thesis
V686
c.2

Vincent
A perfomance measurement methodology for software multiple-backend database systems.